



Microsoft Defender for Devops Report



Yunis Mohammed

Contents

Integrating static analysis tools into the development lifecycle (Defender for DevOps)	2
Introduction	2
Exercise 1: Import the vulnerable code	2
Exercise 2: SAST scan using Bandit locally.....	2
Exercise 3: Configure the Microsoft Security DevOps Azure DevOps extension	4
Exercise 4: Configure the Microsoft Security DevOps GitHub actions	6
Exercise 5: DevOps Security Workbook	7
Conclusion	8

Integrating static analysis tools into the development lifecycle (Defender for DevOps)

Introduction

Static analysis tools are a critical part of the development lifecycle for identifying security vulnerabilities early in the development process. Defender for DevOps is a command line application that integrates static analysis tools into the development lifecycle. It installs, configures, and runs the latest versions of static analysis tools, including Bandit, BinSkim, ESLint, Credscan, Template Analyzer, Terrascan, and Trivy.

In this lab I will be walking through the process of configuring Defender for DevOps in Azure DevOps and GitHub Actions. By the end of the lab, I will be able to:

- Run a SAST scan using Bandit locally.
- Configure the Microsoft Security DevOps Azure DevOps extension.
- Configure the Microsoft Security DevOps GitHub actions.
- Create a DevOps Security Workbook.

Exercise 1: Import the vulnerable code

1. On GitHub, fork the vulnerable code from **S2FrdQ/VulnDjango** ([github.com](https://github.com/S2FrdQ/VulnDjango)).
2. On Azure DevOps, create a new Private project and name it **VulnDjango**. Navigate to Repos, and import the vulnerable code from <https://github.com/S2FrdQ/VulnDjango>.

Exercise 2: SAST scan using Bandit locally.

Bandit - The Bandit is a tool designed to find common security issues in Python code. To do this Bandit, processes each file, builds an AST, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report. Bandit was originally developed within the OpenStack Security Project and later rehomed to PyCQA.

1. Download the source code locally;
git clone https://github.com/S2FrdQ/VulnDjango webappdjango then cd webappdjango

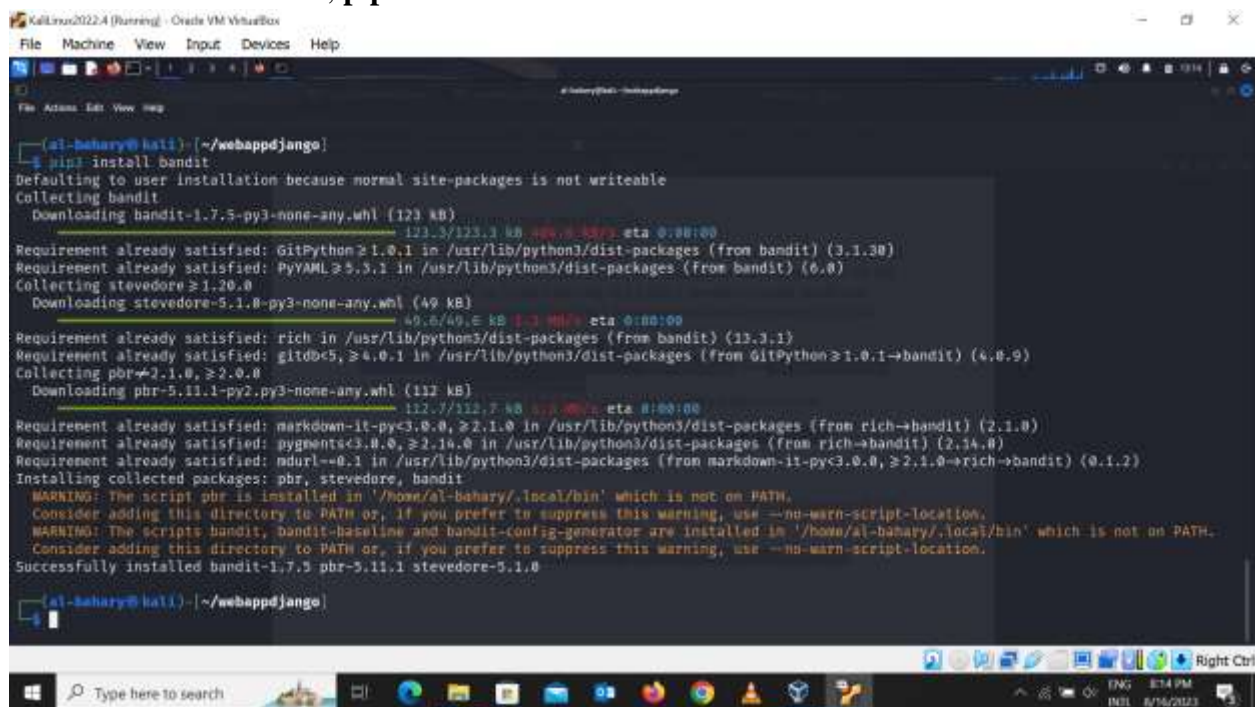


```
KaliLinux20224 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

(al-bahary@kali) ~
$ git clone https://github.com/S2RedQ/VulnDjango webappdjango
Cloning into 'webappdjango' ...
remote: Enumerating objects: 318, done.
remote: Counting objects: 1885 (46/46), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 318 (delta 38), reused 24 (delta 24), pack-reused 272
Receiving objects: 100% (318/318), 1.81 MiB | 776.00 KiB/s, done.
Resolving deltas: 100% (146/146), done.

(al-bahary@kali) ~
$
```

2. Install Bandit; pip3 install bandit



```
KaliLinux20224 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

(al-bahary@kali) ~/webappdjango
$ pip3 install bandit
Defaulting to user installation because normal site-packages is not writeable
Collecting bandit
  Downloading bandit-1.7.5-py3-none-any.whl (123 kB)
    123.3/123.3 kB 400.0 KiB/s eta 0:00:00
Requirement already satisfied: GitPython>=1.0.1 in /usr/lib/python3/dist-packages (from bandit) (3.1.30)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/lib/python3/dist-packages (from bandit) (6.0)
Collecting stevedore>=1.20.0
  Downloading stevedore-5.1.0-py3-none-any.whl (49 kB)
    49.0/49.0 kB 4.3 KiB/s eta 0:00:00
Requirement already satisfied: rich in /usr/lib/python3/dist-packages (from bandit) (13.3.1)
Requirement already satisfied: gltDb<5, >=4.0.1 in /usr/lib/python3/dist-packages (from GitPython>=1.0.1->bandit) (4.0.9)
Collecting pbr>=2.1.0, <=2.0.0
  Downloading pbr-5.11.1-py2.py3-none-any.whl (112 kB)
    112.7/112.7 kB 511.0 KiB/s eta 0:00:00
Requirement already satisfied: markdown-it-py<3.0.0, >=2.1.0 in /usr/lib/python3/dist-packages (from rich->bandit) (2.1.0)
Requirement already satisfied: pygments<3.0.0, >=2.14.0 in /usr/lib/python3/dist-packages (from rich->bandit) (2.14.0)
Requirement already satisfied: mdurl<=0.1 in /usr/lib/python3/dist-packages (from markdown-it-py<3.0.0, >=2.1.0->rich->bandit) (0.1.2)
Installing collected packages: pbr, stevedore, bandit
WARNING: The script pbr is installed in '/home/al-bahary/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The scripts bandit, bandit-baseline and bandit-config-generator are installed in '/home/al-bahary/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed bandit-1.7.5 pbr-5.11.1 stevedore-5.1.0

(al-bahary@kali) ~/webappdjango
$
```

3. If a warning is issued to add Directory to path, add using the below command:

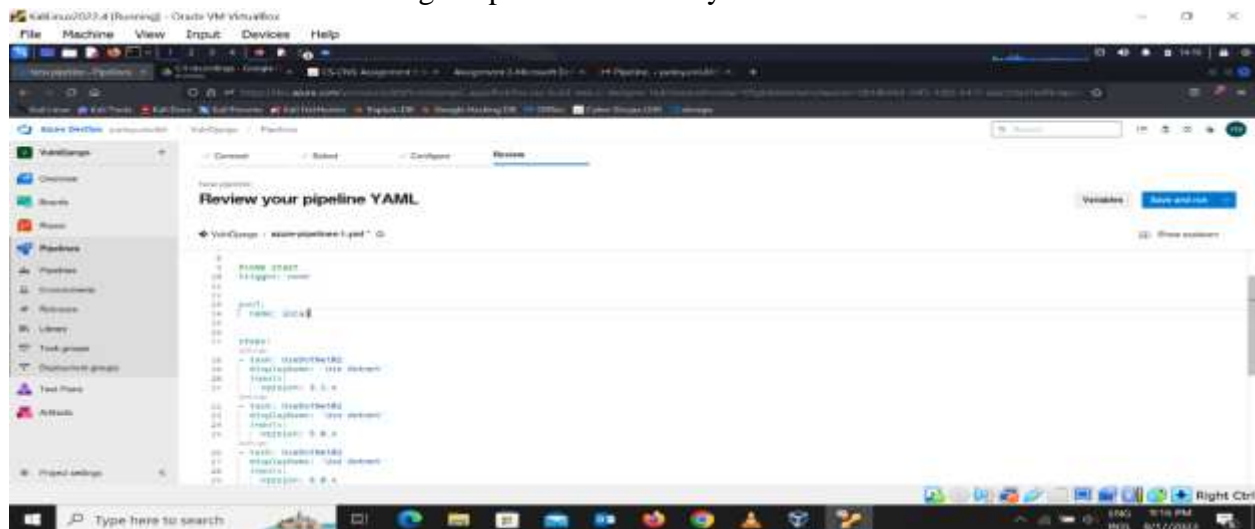
export PATH="/home/kali/.local/bin:\$PATH"

To explore bandit –help.

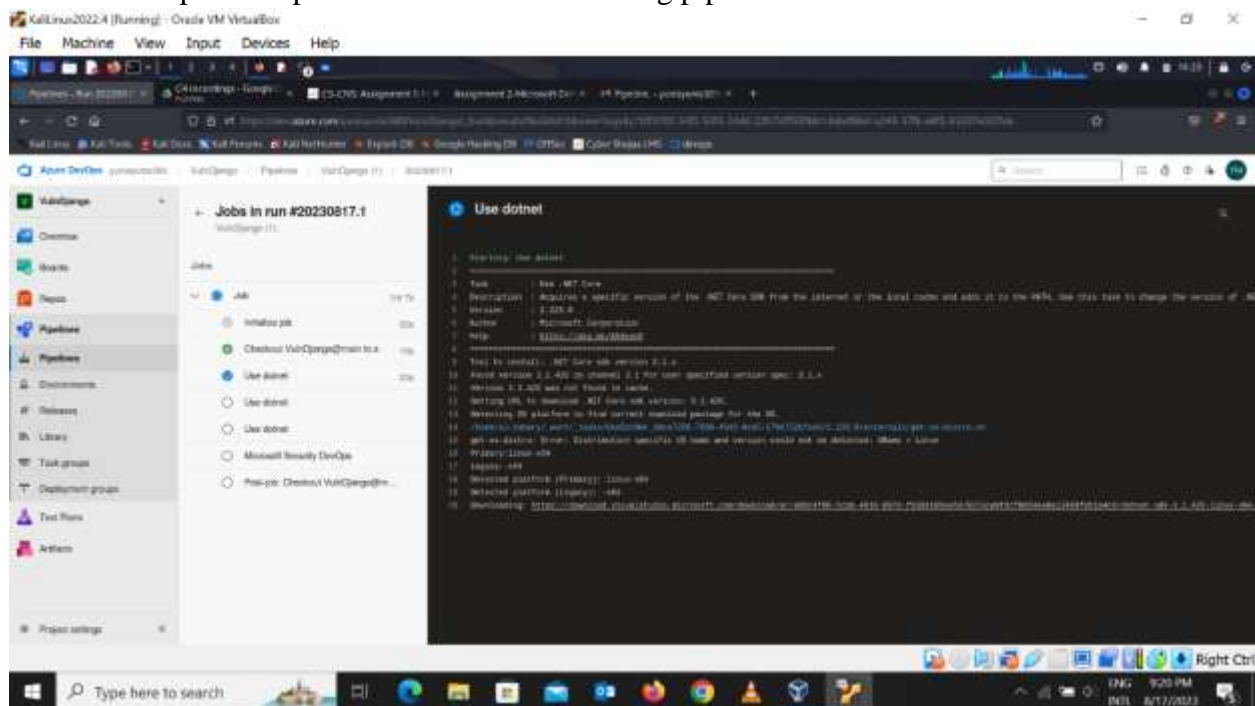
4. Run the scanner - We are using the tee command here to show the output and store it in a file simultaneously. **bandit -r . Basic scan**

bandit -r . -f json | tee bandit-output.json.

6. On the Where is your code? window, select Azure Repos Git (YAML) and select the VulnDjango repository.
7. On Add the following scripts as in into the yaml file.



8. Click Save and run and let the pipeline run. You can check progress by going to Pipeline Pipelines and select the running pipeline.

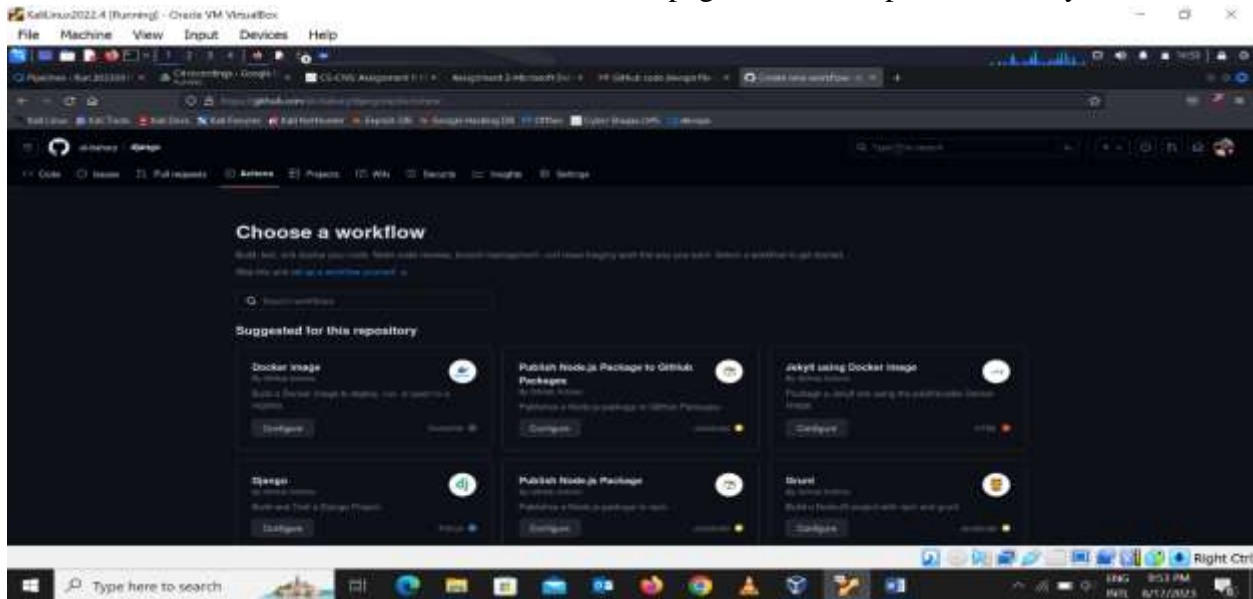


9. When done, you can view security vulnerabilities found by Microsoft Security DevOps, by clicking Scans.

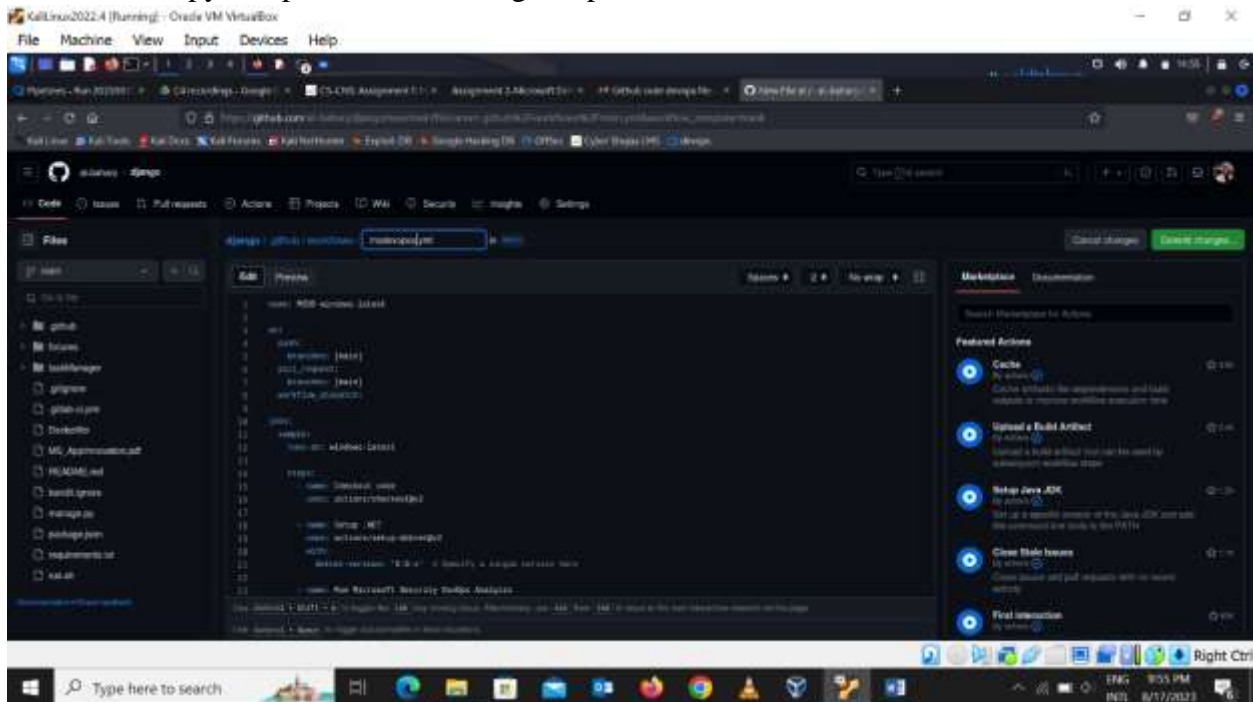
Note: Install the SARIF SAST Scans Tab extension on the Azure DevOps organization in order to ensure that the generated analysis results will be displayed automatically under the Scans tab.

Exercise 4: Configure the Microsoft Security DevOps GitHub actions

1. Navigate to your **VulnDjango** GitHub repo.
2. Select Actions.
3. Select New workflow.
4. On the Get started with GitHub Actions page, select set up a workflow yourself.

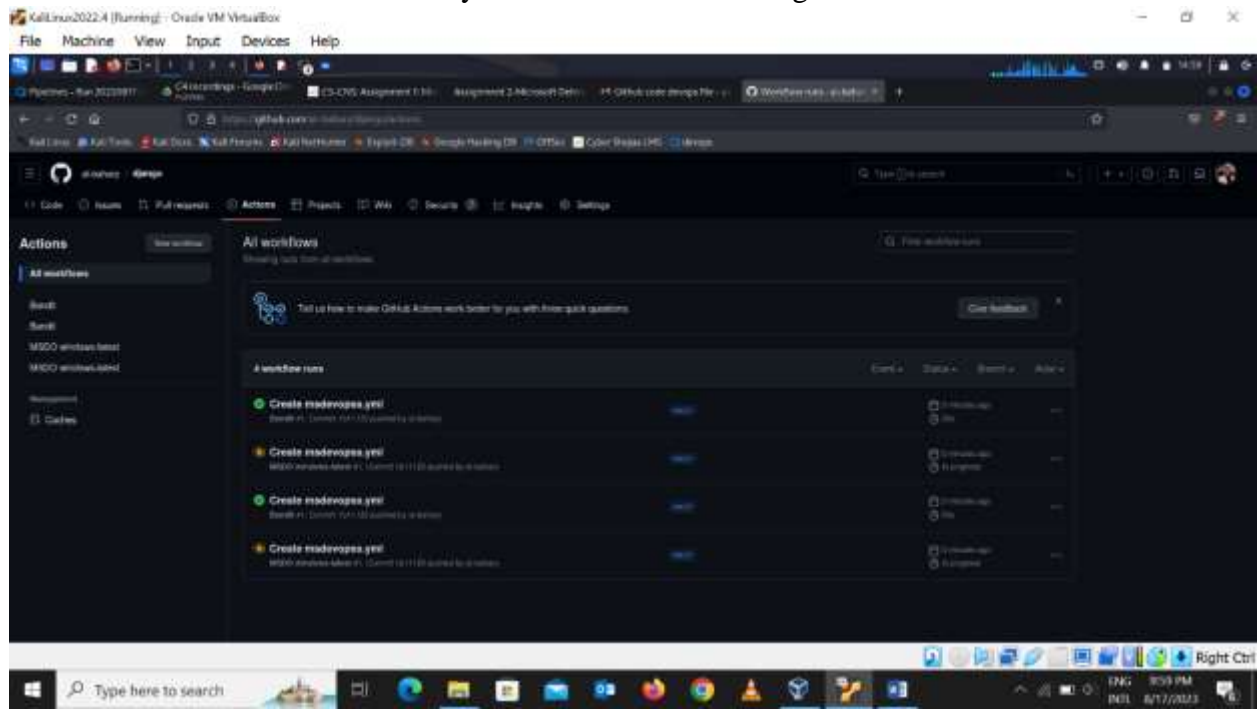


5. In the text box, enter a name for your workflow file. For example, **msdevopssec.yml**.
6. Copy and paste the following sample action workflow into the Edit new file tab.



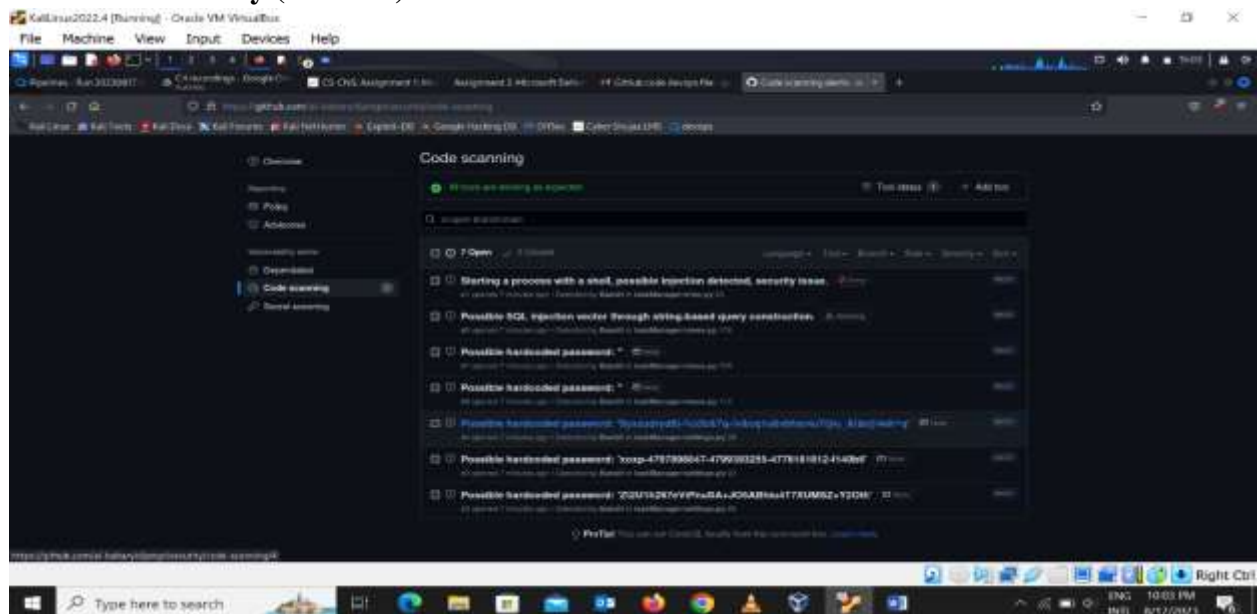
7. For details on various input options, see **action.yml**
8. Select Start commit.
9. Select Commit new file.

10. Select Actions and verify the new action is running.



Exercise 5: DevOps Security Workbook

1. Navigate to **Defender for Cloud**, click on **Workbooks**, then click on **DevOps Security (Preview)** to launch the Workbook.



Conclusion

I have learnt a lot in this lab about the importance of static analysis tools in the development lifecycle and how Defender for DevOps can be used to integrate these tools into the process. I also learned how to configure Defender for DevOps in Azure DevOps and GitHub Actions. One of the key points that I learned is that static analysis tools can help to identify security vulnerabilities early in the development process, when they are easier and cheaper to fix. This is because static analysis tools can scan code for potential vulnerabilities without actually running the code. This can be a valuable tool for preventing security vulnerabilities from making it into production.

I have learnt how to configure Microsoft Security DevOps GitHub actions, which can automate security procedures within GitHub. This is a practical and efficient way to improve security, and it can be easily integrated into existing development workflows. I also created a DevOps Security Workbook, which is a valuable resource for tracking security vulnerabilities and remediation. This workbook will help me to stay organized and ensure that security is always a top priority. Overall, I am confident that I have gained the skills and knowledge necessary to improve the security of applications using GitHub Actions and Microsoft Security DevOps.