

# DevSecOps Pipeline with GitHub Actions

Long Chen, Kaiyu Dong, Shaolun Liu

Simon Fraser University  
`{long.chen.3,kaiyu.dong,shaolun.liu}@sfu.ca`

Supervisor: Mohammad Tayebi, Magno Logan

**Keywords:** DevSecOps · GitHub Action · AWS.

# Contents

DevSecOps Pipeline with GitHub Actions .....	1
<i>Long Chen, Kaiyu Dong, Shaolun Liu</i>	
1 Introduction .....	4
1.1 DevSecOps .....	4
1.1.1 What is DevOps? .....	4
1.1.2 How does DevOps approach security? .....	4
1.1.3 DevSecOps tools .....	5
1.2 GitHub Actions .....	5
1.2.1 Event or Action .....	6
1.2.2 YAML file .....	6
1.2.3 Workflows .....	7
1.2.4 Jobs .....	7
1.2.5 Runners .....	8
2 Workflow Tools .....	8
2.1 CodeQL .....	8
2.1.1 Introduction .....	8
2.1.2 Integration .....	8
2.2 GitGuardian .....	9
2.2.1 Introduction .....	9
2.2.2 Integration .....	9
2.2.3 Configuration .....	11
2.3 Gitleaks .....	11
2.3.1 Introduction .....	11
2.3.2 Integration .....	11
2.3.3 Configuration .....	12
2.4 Semgrep .....	12
2.4.1 Introduction .....	12
2.4.2 Integration .....	13
2.4.3 Configuration .....	14
2.5 Trivy .....	14
2.5.1 Introduction .....	14
2.5.2 Integration .....	14
2.5.3 Configuration .....	16
2.6 Snyk .....	16
2.6.1 Introduction .....	16
2.6.2 Integration .....	16
2.6.3 Configuration .....	17
2.7 Sonar Cloud .....	17
2.7.1 Introduction .....	17
2.7.2 Integration .....	18
2.8 AWS ECR .....	19

2.8.1	Introduction . . . . .	19
2.8.2	Integration . . . . .	20
2.9	AWS ECS . . . . .	21
2.9.1	Introduction . . . . .	21
2.9.2	Integration . . . . .	22
2.10	Nuclei . . . . .	24
2.10.1	Introduction . . . . .	24
2.10.2	Integration . . . . .	24
2.10.3	Configuration . . . . .	25
2.11	OWASP ZAP . . . . .	25
2.11.1	Introduction . . . . .	25
2.11.2	Integration . . . . .	26
2.11.3	Configuration . . . . .	28
	References . . . . .	28

## 1 Introduction

### 1.1 DevSecOps

#### 1.1.1 What is DevOps?

DevOps is a continuous process including eight parts: Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor.



**Fig. 1.** DevOps

The DevOps tools help automate the lifecycle of the application. By using the DevOps tools, processes that previously had to be performed manually and took a lot of time can be executed quickly and continuously. For example, updating code, setting up a new environment, or continuous automated testing of the application.

DevOps combines the software development team and the IT operation team. But where is the security team? Is there a security test in the DevOps test stage? The answer is no. The test stage here means unit test and integration test, not security test.

#### 1.1.2 How does DevOps approach security?

In the DevOps process, security was 'tacked on' to software at the end of the development cycle by a separate security team and was tested by a separate quality assurance team.

But this may cause a problem. If the security team finds some vulnerabilities at the end of the development cycle, fixing the code and security issues may be very difficult, which may lead to huge time delays and may cost a lot of money.

This kind of security problem resulted in the introduction of DevSecOps.

### 1.1.3 DevSecOps tools

DevSecOps is an augmentation of DevOps to allow security practices to be integrated into the DevOps approach. Security practices and testing are performed earlier in the development lifecycle. Security is tested in three main areas: static, software composition, and dynamic.

There are four kinds of DevSecOps tools in the three areas:

- **Software Composition Analysis (SCA)**  
Software Composition Analysis is an application security methodology for managing open source components. Using SCA, development teams can quickly track and analyze any open-source component brought into a project. SCA tools can discover all related components, their supporting libraries, and their direct and indirect dependencies. SCA tools can also detect software licenses, and deprecated dependencies, as well as vulnerabilities, and potential exploits. The scanning process generates a bill of materials (BOM), providing a complete inventory of a project's software assets.  
Our project used Snyk as the SCA tool.
- **Static Application Security Testing (SAST)**  
SAST tools focus on the code content of the application, which is white-box testing. A SAST tool scans the source code of applications to identify potential security vulnerabilities. Our project used Sonar, CodeQL, and Trivy as SAST tools.  
There is one significant difference between SCA and SAST. While SCA identifies vulnerabilities in open-source code, SAST detects vulnerabilities in proprietary code.  
Our project used Sonar, CodeQL, and Trivy as SAST tools.
- **Dynamic Application Security Testing (DAST)**  
DAST tools perform a black-box test in a running application. Unlike SCA and SAST tools, DAST tools do not have access to the source code and therefore detect vulnerabilities by performing attacks, such as SQL injections, OS command injections, cross-site scripting, etc.  
Our project used Nuclei and ZAP as DAST tools.
- **Secrets Scanning**  
Secret scanning tools scan the source code for known types of secrets in order to prevent leaking secrets, such as passwords, tokens, private keys, etc. Actually, it can be regarded as a kind of SAST tool. The reason it is listed separately is that other SAST tools focus on vulnerabilities in the code, while secret scanning tools only focus on secrets.  
Our project used Gitleaks and GitGuardian as the Secrets Scanning tool.

## 1.2 GitHub Actions

GitHub Action was launched in 2018 as a CI/CD platform-native automation tool. CI/CD stands for continuous integration and continuous delivery. It is a platform that allows developers to automate the build, test, and pipeline deployment right next to their code on GitHub. Developers can create workflow at

every pull request of the repository or merge to the branches. It is compatible with Linux, Windows, and macOS virtual machines and your self-hosted runners in the data center or cloud infrastructure.

### 1.2.1 Event or Action

Event is what happens to or in the repository. It could be a pull request, merge or even synchronize the code to any branch. The GitHub action enables users to create software development lifecycle workflows directly in the repository.

Different tasks, also called actions, make up workflows that can run automatically and are triggered by certain events. It can perform a complex but frequently repeated task, which helps with the redundant work of writing workflow files. For instance, an action can pull the code from the GitHub repository and set up an environment or authentications.

### 1.2.2 YAML file

To understand the workflow and actions, users need to understand the logic of the YAML file, which defines the actions or runners and their specifications. Then, it is straightforward for users to interpret and modify the files and variables, which can be deployed across different languages and easily mapped into a dictionary. The YAML file is not a Markup Language but a serialization language that is getting increasingly popular in recent years. It is widely used for configurations. The object serialization abilities make it a viable replacement for languages like JSON. YAML is short for Yet Another Markup Language.

```

1 name: DevSecOps Pipeline
2 on:
3   push:
4     branches: [ "master" ]
5   pull_request:
6     branches: [ "master" ]
7
8 env:
9   AWS_REGION: us-west-2
10  ECR_REPOSITORY: jpetstore
11  ECS_SERVICE: jpetstore-service
12  ECS_CLUSTER: jpetstore-cluster
13  ECS_TASK_DEFINITION: aws/task_definition.json
14  CONTAINER_NAME: sample-app
15

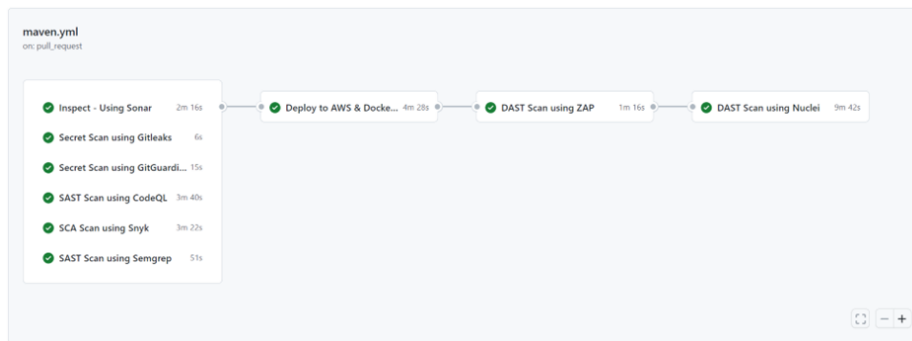
```

The attached image shows a portion of the YAML file in this project. As defined in the file, once the new commit is pushed to the master branch or the pulling request is executed, the workflow will start. Users can customize their own environments and jobs.

### 1.2.3 Workflows

It is a configuration that regulates either running one or more jobs. It is defined in a YAML file with keys and values. It can be triggered by the events on the repository, manually, or at a limited schedule. Workflow is always defined in the ".github/workflow" directory in the user's repository. Users can have multiple workflows defined simultaneously; each workflow can hold different tasks. For instance, users can have one workflow for building and pulling the request and another to deploy the application each time a new release is generated.

Users must write YAML files in the workflow directory and store different workflows separately. After the user commits their code to GitHub, the workflow will run as defined. Under the Actions tab on GitHub, the running process is displayed as a list. Please see the Figure: "Workflow in Action Tab." In this case, the jobs, such as Sonar, GitLeaks, GitGuardian, etc., are running parallelly. And they are set as dependencies regarding the Trivy.



**Fig. 2.** Workflow in Action Tab

The user can see the steps that make up this job by clicking on each job. The Figure: "Steps Example" explains the details of the Sonar scanner. In each job, the steps are running in order.

### 1.2.4 Jobs

Each job is constructed by a series of tasks or steps that happens when a runner is triggered. It is Either running an action or executing a shell script. Users can share data from one step to another if they are included in a single job. For instance, once the application is built, the user may test the application in the following step. By the default setting, jobs are running parallelly or simultaneously when executing the workflow. Still, users are allowed to set the dependencies of each job, which makes the workflow architecture allow one job to run after a specific job.

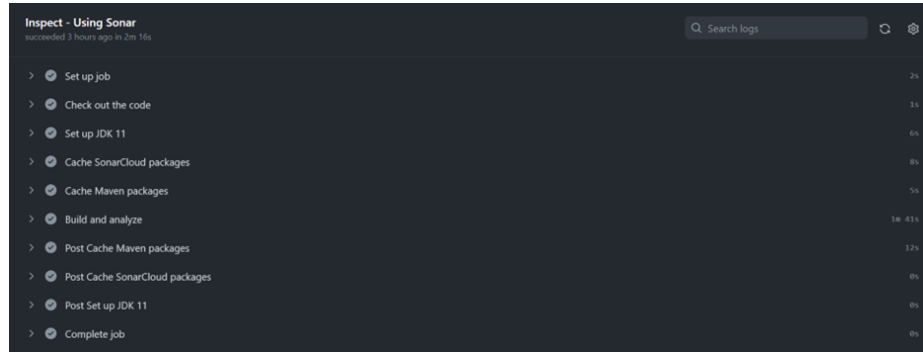


Fig. 3. Steps Example

### 1.2.5 Runners

GitHub provides multiple operating systems, such as Ubuntu Linux, Microsoft Windows and macOS, to run the workflows. The runner is the server that runs the workflow when the user commits. Each time when the user executes the workflow, the runner will start as a freshly provided virtual machine. User can also host their own runners if it requires a specific operating system or specific hardware configuration.

## 2 Workflow Tools

### 2.1 CodeQL

#### 2.1.1 Introduction

CodeQL is the analysis engine focused on security checks, such as vulnerabilities, bugs, and other errors, as well as queries against the database. Other researchers and developers write CodeQL queries that users can implement. Meanwhile, users can modify it by themselves.

Security engineers can identify vulnerabilities by implementing variant analysis. It is a process of using known vulnerabilities as a seed to detect more potential and similar vulnerabilities. Such a technique can ensure these security issues are appropriately fixed across different codebases.

#### 2.1.2 Integration

Deploying CodeQL is the most efficient way to perform variant analysis. Users can easily detect some vuln by implementing standard CodeQL queries. Users can also build their queries to find logical variants of the same bugs the traditional scan could miss.

```

1  CodeQL-Build:
2    name: SAST Scan using CodeQL
3    runs-on: ubuntu-latest

```



```

4     permissions:
5         security-events: write
6         actions: read
7         contents: read
8     steps:
9         - name: Checkout repository
10           uses: actions/checkout@v3
11         - name: Initialize CodeQL
12           uses: github/codeql-action/init@v2
13         - name: Autobuild
14           uses: github/codeql-action/autobuild@v2
15         - name: Perform CodeQL Analysis
16           uses: github/codeql-action/analyze@v2

```

Please see above to dive into the CodeQL YAML code. Users can deploy the scan by first preparing the code for the scan and creating the CodeQL database. The second step is to run the QodeQL queries against the database and finally interpret or analyze the result.

## 2.2 GitGuardian

### 2.2.1 Introduction

GitGuardian Shield is a Command Line Interface application that can run on the local environment or continuous integration environment to help developers to detect more than 200 types of secrets, as well as some other potential security vulnerabilities and policy breaks. It uses a public API based on py-GitGuardian to scan the files and detect potential secrets or security issues. The workflow will be marked as failed if the scan detects any secret leaks in the commit.

### 2.2.2 Integration

The attached image is a YAML file of GitGuardian deployment. Users need to acquire API Token from the GitGuardian server by registering their own account and adding it to GitHub. in the YAML file; users need to first checkout the repository. Therefore, the new workflow can run the scan. Once the scan completes, the result can either be stored in GitHub or automatically reserved in GitGuardian Server. Log into the GitGuardian account will let the user access their scan result. The next image shows the scan result.

```

1     GitGuardian:
2         name: Secret Scan using GitGuardian
3         runs-on: ubuntu-latest
4         steps:
5             - name: Checkout
6               uses: actions/checkout@v2
7               with:
8                 fetch-depth: 0

```

```

9       - name: GitGuardian scan
10         uses: GitGuardian/ggshield-action@master
11         with:
12           args: -v --all-policies
13         env:
14           GITHUB_PUSH_BEFORE_SHA: ${ github.event.before }}
15           GITHUB_PUSH_BASE_SHA: ${ github.event.base }}
16           GITHUB_PULL_BASE_SHA: ${ {
17     ↪ github.event.pull_request.base.sha }}
18           GITHUB_DEFAULT_BRANCH: ${ {
19     ↪ github.event.repository.default_branch }}
20           GITGUARDIAN_API_KEY: ${ secrets.GITGUARDIAN_API_KEY }}

```

### Occurrences

DATE	WHO	WHERE	PRESENC	TAGS
December 15th, 2... 16:58	Kaiyu Dong 32695019+dky815...	cl456852/se... 37513e4 test.yml		Exposed publi... Test file

1 - 1 of 1

cl456852/secure\_flow
 37513e4
 test.yml

GENERIC HIGH ENTROPY SECRET **apikey**

```

@@ -0,0 +1 @@
1 + discord_client_secret: '8dyfuiRyq=vVc3RRr-edRk-fk__J1tpz'
0 2 No newline at end of file

```

**Fig. 4.** GitGuardian Scan Result

Users can find the errors detected from the result page in GitGuardian Server. In the sample case, There are secret leaks detected, which means the keys are not appropriately hidden, such as storing them in the "Secret" folder on Github. It is risky for developers to publish their tokens, secret keys, and code on any platform.

Users may edit information regarding the error on the right-hand side of the scan result, such as assigning an assignee to handle the incident or setting the issue's severity. In addition, the user may choose to ignore the incident if it is less severe. Finally, recommendations on how to fix the problem are also provided in this section.

### 2.2.3 Configuration

Users need to register their GitGuardian account to acquire Token and link the project and repository of GitHub to their GitGuardian account to make it active and store the scan each time they commit their code.

## 2.3 Gitleaks

### 2.3.1 Introduction

Gitleaks is an open-source tool that is used to scan Git repositories for sensitive information. It can be used by organizations to identify and fix potential security vulnerabilities in their repositories and can be run locally or as a cloud-based service.

Gitleaks is designed to search deep within a repository, looking for sensitive information that may have been accidentally committed. This includes things like passwords, private keys, and other confidential data. If Gitleaks finds anything, it will alert the user and provide guidance on how to fix the issue.

Gitleaks is a valuable tool for organizations that rely on Git for version control and collaboration, as it helps them ensure that sensitive information is not accidentally exposed through their repositories. It is also useful for developers who want to ensure the security of their own repositories.



### 2.3.2 Integration

To use the “gitleaks-action” with your own Github project, you will need to create a new workflow file in your repository and add a job that uses the gitleaks-action. Here is an example YAML file that runs Gitleaks on all pull requests and pushes:



```

1  name: gitleaks
2  on:
3    pull_request:
4    push:
5  jobs:
6    scan:
7      name: gitleaks
8      runs-on: ubuntu-latest
9      steps:
10       - uses: actions/checkout@v3
11         with:
12           fetch-depth: 0
13       - uses: gitleaks/gitleaks-action@v2
14         env:
15           GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Once you have added the job to your workflow file, commit it to your repository and push the changes to Github. The workflow will automatically be triggered when you create a new pull request or push to the repository.

<b>Artifacts</b> Produced during runtime							
Name	Size						
 gitleaks-results.sarif	40.2 KB						

<b>Secret Scan using Gitleaks summary</b> <span>...</span>							
 <b>Gitleaks detected secrets</b> 							
Rule ID	Commit	Secret URL	Start Line	Author	Date	Email	File
discord-client-secret	<a href="#">37513e4</a>	<a href="#">View Secret</a>	2	Kaiyu Dong	2022-12-16T00:58:36Z	32695019+dky815@users.noreply.github.com	<a href="#">test.yml</a>

Job summary generated at run-time

**Fig. 5.** Gitleaks Scan Results

### 2.3.3 Configuration

- **GITHUB\_TOKEN:**  
This variable is automatically assigned by GitHub when any action gets kicked off. You can read more about the token here. gitleaks-action uses this token to call a GitHub API to comment on PRs.
- **GITLEAKS\_LICENSE** (required for organizations, not required for user accounts):  
A gitleaks-action license can be obtained at gitleaks.io. It should be added as an encrypted secret to the repo or to the organization.
- **Who need a license key?**  
If you are scanning repos that belong to an organization account, you will need to obtain a license key. You can obtain a free "Trial" license key for scanning unlimited repos for up to 2 weeks. Scanning repos after the trial license has expired requires a paid license.  
If you are scanning repos that belong to a personal account, then no license key is required.
- **How to get a license key?**  
You can visit gitleaks.io to sign up for a free trial license key for unlimited repo scanning for up to 2 weeks, or choose from a paid tier to enable scanning after the trial license has expired.
- **How to use a custom gitleaks configuration?**  
This GitHub Action follows a similar order of precedence as the gitleaks CLI tool. You can use GITLEAKS\_CONFIG to explicitly set a config path or create a gitleaks.toml at the root of the repo which will be automatically detected and used by gitleaks-action.

## 2.4 Semgrep

### 2.4.1 Introduction

Semgrep is an open-source static analysis tool that can be used to find and fix

security vulnerabilities and code defects in your codebase. It is designed to be fast, flexible, and easy to use, and can be run on a wide range of programming languages.

Semgrep uses a set of customizable rules to scan your code for potential issues, and it provides clear, actionable feedback on how to fix any problems it finds.

Semgrep is particularly useful for organizations that want to ensure the security and quality of their codebase. It can be run as part of a continuous integration pipeline to catch issues early in the development process, and it can be easily integrated into a variety of tools and platforms.

In addition to its security and code quality features, Semgrep is also useful for finding and fixing formatting issues, refactoring code, and enforcing coding standards. It is a powerful tool that can help developers write cleaner, more maintainable code.

#### 2.4.2 Integration

To add a Semgrep configuration file in your GitHub Actions pipeline:

Create a `semgrep.yml` file in `.github/workflows` in the repository you want to scan.

Copy the relevant code snippet provided in Sample GitHub Actions configuration file.

Paste the relevant code snippet to `semgrep.yml` file. This is your Semgrep configuration file for GitHub Actions.

Commit the configuration file under `/REPOSITORY-ROOT-DIRECTORY/.github/workflows/semgrep.yml`.

The Semgrep job starts automatically upon detecting the committed `semgrep.yml` file.

Here is a sample YAML file of integrating Semgrep into Github Actions:

```

1  semgrep:
2    name: SAST Scan using Semgrep
3    runs-on: ubuntu-latest
4
5    container:
6      image: returntocorp/semgrep
7
8    if: (github.actor != 'dependabot[bot]')
9    steps:
10     - uses: actions/checkout@v3
11     - run: semgrep ci --sarif --output=semgrep.sarif
12       env:
13         SEMGREP_RULES: p/default
14     - name: Upload SARIF file to GitHub Security tab
15       uses: github/codeql-action/upload-sarif@v2
16       with:

```

```

17     sarif_file: semgrep.sarif
18     if: always()

```

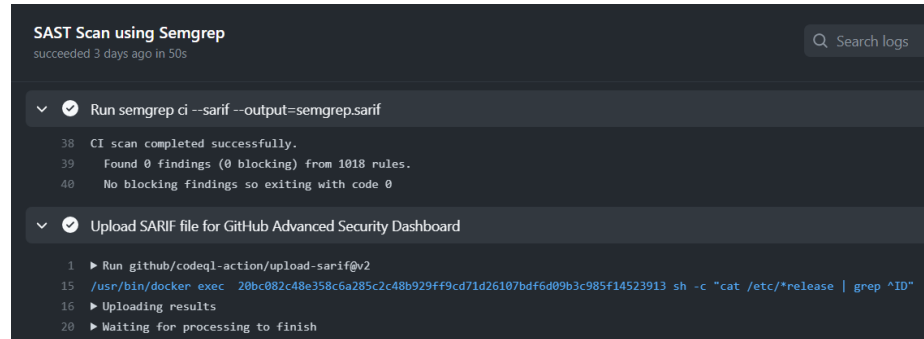


Fig. 6. Semgrep Scan Results

### 2.4.3 Configuration

You do not need to set additional configurations.

## 2.5 Trivy

### 2.5.1 Introduction

Trivy (short for "Trivial Vulnerability Examiner") is an open-source tool that can be used to scan container images for vulnerabilities. It is designed to be fast and easy to use, and can be run on a wide range of platforms, including Linux, macOS, and Windows.

Trivy will perform a deep scan of the docker image, looking for known vulnerabilities in the packages and libraries it uses. If it finds any vulnerabilities, it will alert you and provide guidance on how to fix the issue.

Trivy is particularly useful for organizations that use containerization in their development and deployment processes, as it can help them ensure the security of their container images. It is also a useful tool for developers who want to ensure that their container images are secure before deploying them to production environments.

In addition to its security features, Trivy is also useful for identifying outdated packages and libraries in container images, and for finding and fixing other issues that can affect the performance and stability of containerized applications.

### 2.5.2 Integration

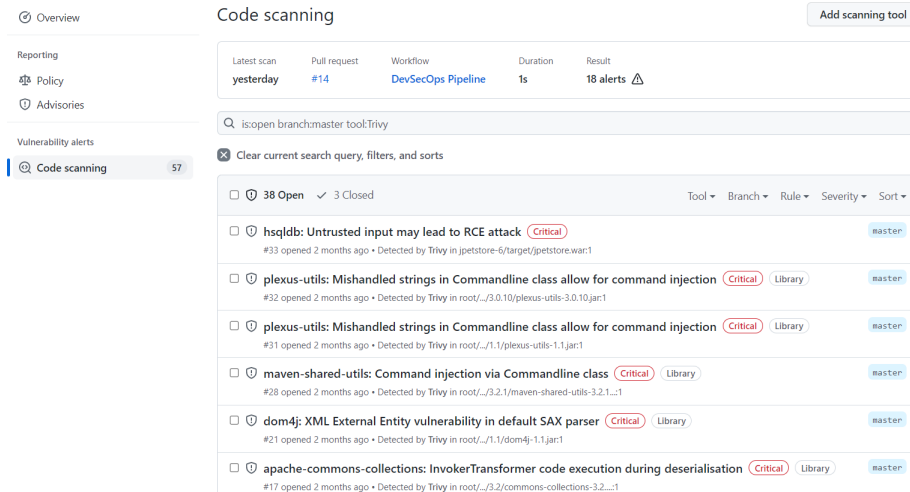
To use the trivy-action with your own Github project, you will need to create a new workflow file in your repository and add a job that uses the trivy-action.

Here is a sample YAML file of integrating Trivy into Github Actions:

```

1 Trivy:
2   name: Image scan by Trivy
3   runs-on: ubuntu-20.04
4   steps:
5     - name: Checkout code
6       uses: actions/checkout@v3
7
8     - name: Build an image from Dockerfile
9       run: |
10         docker build -t ContainerRegistry:${{ github.sha }} .
11
12     - name: Run Trivy vulnerability scanner
13       uses: aquasecurity/trivy-action@master
14       with:
15         image-ref: 'ContainerRegistry:${{ github.sha }}'
16         format: 'sarif'
17         output: 'trivy-results.sarif'
18
19     - name: Upload Trivy scan results to GitHub Security tab
20       uses: github/codeql-action/upload-sarif@v2
21       with:
22         sarif_file: 'trivy-results.sarif'

```



**Code scanning** Add scanning tool

Latest scan	Pull request	Workflow	Duration	Result
yesterday	#14	DevSecOps Pipeline	1s	18 alerts <a href="#">△</a>

is:open branch:master tool:Trivy

[Clear current search query, filters, and sorts](#)

☐ 38 Open ☒ 3 Closed Tool Branch Rule Severity Sort

<input type="checkbox"/> <a href="#">hsqldb: Untrusted input may lead to RCE attack</a> <span>Critical</span> <span>Library</span> <span>master</span>	#33 opened 2 months ago • Detected by Trivy in jpetstore-6/target/jpetstore.war:1
<input type="checkbox"/> <a href="#">plexus-utils: Mishandled strings in Commandline class allow for command injection</a> <span>Critical</span> <span>Library</span> <span>master</span>	#32 opened 2 months ago • Detected by Trivy in root/.../3.0.10/plexus-utils-3.0.10.jar:1
<input type="checkbox"/> <a href="#">plexus-utils: Mishandled strings in Commandline class allow for command injection</a> <span>Critical</span> <span>Library</span> <span>master</span>	#31 opened 2 months ago • Detected by Trivy in root/.../1.1/plexus-utils-1.1.jar:1
<input type="checkbox"/> <a href="#">maven-shared-utils: Command injection via Commandline class</a> <span>Critical</span> <span>Library</span> <span>master</span>	#28 opened 2 months ago • Detected by Trivy in root/.../3.2.1/maven-shared-utils-3.2.1...1
<input type="checkbox"/> <a href="#">dom4j: XML External Entity vulnerability in default SAX parser</a> <span>Critical</span> <span>Library</span> <span>master</span>	#21 opened 2 months ago • Detected by Trivy in root/.../1.1/dom4j-1.1.jar:1
<input type="checkbox"/> <a href="#">apache-commons-collections: InvokerTransformer code execution during deserialisation</a> <span>Critical</span> <span>Library</span> <span>master</span>	#17 opened 2 months ago • Detected by Trivy in root/.../3.2/commons-collections-3.2...1

Fig. 7. Trivy Scan Results

### 2.5.3 Configuration

To scan docker image from your container registry with Trivy's built-in image scan, all you have to do is to set the target of your container registry.

## 2.6 Snyc

### 2.6.1 Introduction

Snyk is a software company that provides tools for identifying and fixing vulnerabilities in open source software. The company's main product is a cloud-based platform that helps developers and IT professionals secure their applications by continuously scanning for known vulnerabilities in the dependencies and packages used by the application.

Snyk's platform integrates with a variety of development tools and environments, including the command line, Git, Jenkins, and popular IDEs like Visual Studio and Eclipse. It also provides integrations with cloud platforms like AWS and Azure, as well as container orchestration platforms like Kubernetes.

Using Snyk's platform, developers can identify vulnerabilities in their dependencies and packages, and then either fix the vulnerabilities directly in their code or use Snyk's remediation advice to patch the vulnerabilities. In addition to its vulnerability scanning and remediation capabilities, Snyk also offers features for managing open source licenses and for tracking the use of open source components in an organization's software.

### 2.6.2 Integration

Here's an example of how you might set up a workflow that uses the Snyk GitHub Action to scan your repository for vulnerabilities:

```

1  snykScan:
2    name: SCA Scan using Snyk
3    runs-on: ubuntu-latest
4
5    steps:
6      - uses: actions/checkout@v2
7        with:
8          fetch-depth: 0
9      - name: Set up Maven
10        run: mvn -N io.takari:maven:wrapper -Dmaven=3.8.2
11      - name: Run Snyk to check for vulnerabilities
12        continue-on-error: true
13        uses: snyk/actions/maven-3-jdk-11@master
14        env:
15          SNYK_TOKEN: ${ secrets.SNYK_TOKEN }
16        with:
17          args: --sarif-file-output=snyk.sarif
18      - name: Upload result to GitHub Code Scanning

```



```

19     uses: github/codeql-action/upload-sarif@v2
20     with:
21       sarif_file: snyk.sarif

```

### 2.6.3 Configuration

To integrate Snyk into your GitHub Actions workflow, you will need to sign up for a Snyk account and obtain a `SNYK_TOKEN`. Once you have your token, you can use the Snyk GitHub Action to scan your repository for vulnerabilities and receive alerts when vulnerabilities are found.

## 2.7 Sonar Cloud

### 2.7.1 Introduction

SonarCloud is a cloud-based version of SonarQube, an open-source platform for continuous inspection of code quality. It allows developers to analyze the quality of their code and identify issues, such as bugs, vulnerabilities, and security hotspots, in real time as they write code.

SonarCloud supports various programming languages, including Java, C#, C/C++, and JavaScript, and integrates with popular development tools, such as GitHub, Azure DevOps, and Jenkins. It provides a rich set of features to help developers improve the quality of their code, including:

**Static code analysis:** SonarCloud performs a static code analysis to identify issues based on a set of rules, such as coding standards, best practices, and security standards. It also provides automated code review feedback and suggests fixes for identified issues.

**Code coverage:** SonarCloud measures the code coverage of unit tests and helps developers ensure that their code is thoroughly tested. It can also generate test reports and test execution statistics.

**Code duplication:** SonarCloud detects and reports on code duplication, helping developers avoid writing redundant or unnecessarily complex code. It can also generate a report of duplicated blocks of code.

**Code complexity:** SonarCloud measures the complexity of code and helps developers write maintainable and easy-to-understand code. It can generate a report of the most complex files and functions in the codebase.

SonarCloud also provides a web-based user interface that allows developers to view the results of code analysis, track progress over time, and collaborate with their teams. It also provides APIs and integrations with other tools, such as issue trackers, to allow developers to automatically create and track issues identified by SonarCloud.

Overall, SonarCloud is a powerful tool for developers to improve the quality and maintainability of their code. It helps developers identify and fix issues early in the development process, reducing the cost and effort required to fix issues later on. You can learn more about SonarCloud and its features on the SonarCloud website.

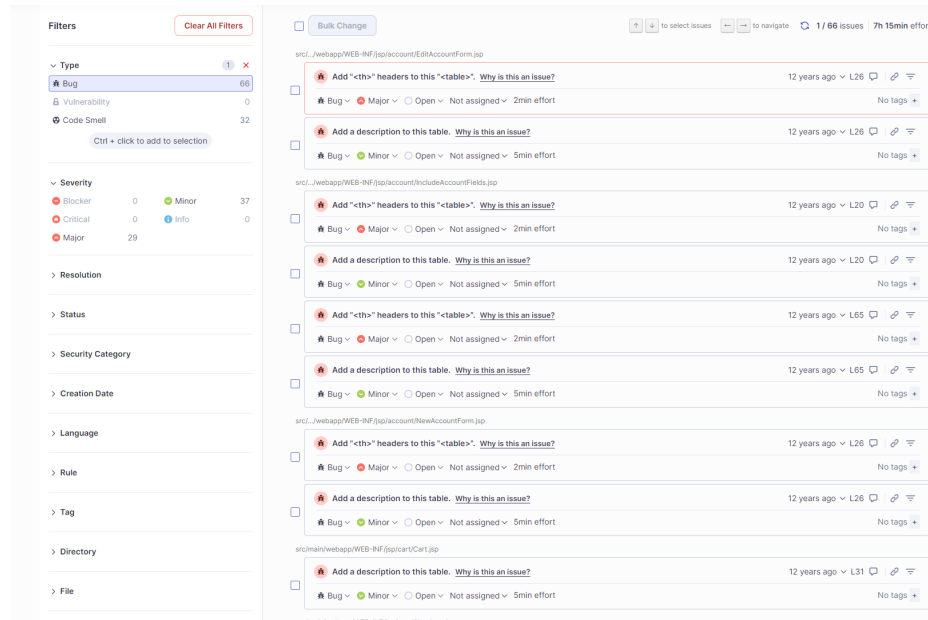


Fig. 8. SonarCloud

### 2.7.2 Integration

To integrate SonarCloud into a GitHub Action using LaTeX, you will need to follow these steps:

First, make sure you have a SonarCloud account and a repository set up on GitHub.

Next, you'll need to create a `.github/workflows` directory in your repository, if it doesn't already exist.

Within the workflows directory, create a new file called `sonarcloud.yml` and add the following YAML code to it:

```

1 on: [push]
2
3 name: SonarCloud
4
5 jobs:
6   build:
7     runs-on: ubuntu-latest
8     steps:
9       - uses: actions/checkout@v2
10      - name: Set up JDK 1.8
11        uses: actions/setup-java@v1
12        with:
13          java-version: 1.8

```

```

14     - name: Build with Maven
15       run: mvn clean package
16     - name: Analyze with SonarCloud
17       uses: sonarsource/sonarcloud-github-action@master
18       env:
19         GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
20         SONAR_TOKEN: ${ secrets.SONAR_TOKEN }

```

This code will trigger the action every time you push a change to your repository, and it will run a series of steps to build and analyze your code with SonarCloud.

Next, you'll need to create a personal access token for SonarCloud and add it to your repository as a secret. To do this, go to your SonarCloud account settings and click on "Security." From there, you can generate a new token and copy it to your clipboard.

In your GitHub repository, go to the "Settings" tab and click on "Secrets." Click on the "New secret" button and give your secret the name "SONAR\_TOKEN." Paste the personal access token you copied from SonarCloud into the value field and click "Add secret."

You should now be all set to run the action and analyze your code with SonarCloud. To do this, commit and push your code to your repository. This will trigger the action, and you should see it running in the "Actions" tab of your repository.

Once the action has been completed, you can view the results of the analysis in the SonarCloud dashboard. From there, you can see a detailed breakdown of your code quality, including issues and potential bugs, as well as any relevant documentation and suggestions for improvement.

## 2.8 AWS ECR

### 2.8.1 Introduction

Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images. ECR is integrated with Amazon ECS, making it easy to deploy containers to ECS without having to manage infrastructure. ECR is also integrated with other AWS services, such as AWS CodePipeline, AWS CodeBuild, and AWS Fargate, which makes it easy to build and deploy containerized applications using a continuous integration and delivery (CI/CD) workflow.

Some key features of ECR include:

**Secure:** ECR uses AWS Identity and Access Management (IAM) to control access to images, so you can grant access to specific users or groups.

**Scalable:** ECR can store and manage millions of images, and is designed to handle high levels of traffic and scale to meet the needs of your applications.

**Low cost:** ECR charges a low rate for storing and serving images, and there are no upfront costs or fees for using the service.

ECR is a valuable tool for developers who want to use Docker to build and deploy applications in the cloud. It provides a simple, scalable, and secure way to store and manage Docker images, and is integrated with other AWS services to enable a seamless CI/CD workflow.

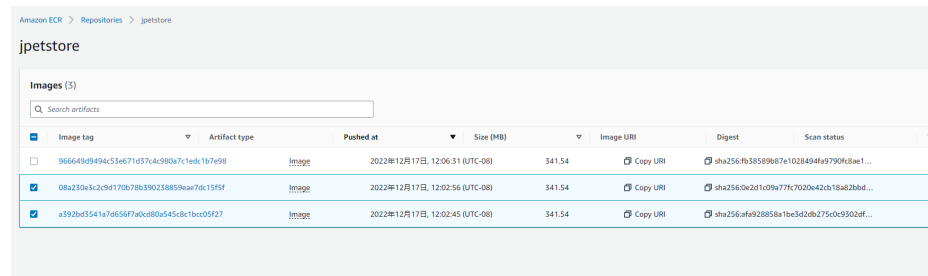


Fig. 9. AWS ECR

### 2.8.2 Integration

To integrate AWS Elastic Container Registry (ECR) into a GitHub Action, you can use the AWS CLI (Command Line Interface) and the `aws ecr` command in a script within the Action. Here is an example of how you can use the AWS CLI to authenticate with ECR and push a Docker image to ECR from a GitHub Action:

First, you will need to create an IAM (Identity and Access Management) user or use an existing IAM user that has permission to access ECR. You will need the access key ID and secret access key for this user.

In your GitHub repository, create a new Action by creating a new file in the `.github/workflows` directory. You can use the following YAML code as a starting point for your Action:

```

1 name: Push Docker image to ECR
2 on: [push]
3 jobs:
4   build-and-deploy:
5     runs-on: ubuntu-latest
6     steps:
7       - uses: actions/checkout@v2
8       - name: Login to AWS ECR
9         run: |
10           aws configure set aws_access_key_id ACCESS_KEY_ID
11           aws configure set aws_secret_access_key SECRET_ACCESS_KEY
12           aws configure set default.region REGION
13           # Replace the ACCOUNT_ID with your AWS account ID

```

```

14     $(aws ecr get-login --no-include-email --registry-ids
    ↪ ACCOUNT_ID)
15   - name: Build and push Docker image to ECR
16     run: |
17         docker build -t REPOSITORY_NAME:TAG .
18         docker push
    ↪ ACCOUNT_ID.dkr.ecr.REGION.amazonaws.com/REPOSITORY_NAME:TAG
19

```

Replace the placeholders in the YAML code with the appropriate values for your environment. In particular, you will need to provide the access key ID and secret access key for your IAM user, the region where your ECR repository is located, and the name of your ECR repository and the tag for your Docker image.

Commit and push your changes to the repository. The Action will be triggered when you push to the repository, and it will run the script to authenticate with ECR and push the Docker image to your ECR repository.

For more information about using the AWS CLI and the `aws ecr` command, you can refer to the AWS documentation or online resources such as the AWS CLI documentation and AWS ECR documentation. You can also find more examples and guidance on using the AWS CLI with GitHub Actions in the GitHub documentation.

## 2.9 AWS ECS

### 2.9.1 Introduction

**Amazon Elastic Container Service (ECS)** is a fully managed container orchestration service provided by Amazon Web Services (AWS). It allows you to run and manage Docker containers on a cluster of Amazon Elastic Compute Cloud (EC2) instances or AWS Fargate, a serverless compute engine for containers.

With ECS, you can easily deploy and run applications in containers without the need to worry about infrastructure management. You can specify the number and type of EC2 instances or Fargate instances to use, and ECS will automatically scale the number of instances up or down based on the workload. You can also use ECS to manage the networking, security, and availability of your containers.

ECS supports various features and integration points, such as:

- **Task definitions:** You can define the containers, CPU, memory, and other resources that your tasks require in a task definition.
- **Service discovery:** You can use Amazon Route 53 to create a private DNS hostname for your tasks, allowing you to easily discover and access the tasks within your VPC (Virtual Private Cloud).
- **Load balancing:** You can use an Amazon Elastic Load Balancer or an Application Load Balancer to distribute traffic across your tasks.

- **Monitoring and logging:** You can use Amazon CloudWatch to monitor the performance and health of your tasks, and you can use AWS CloudTrail to log ECS API calls.
- **Integration with other AWS services:** You can use ECS with other AWS services, such as Amazon ECR (Elastic Container Registry) for container image storage, Amazon EC2 Auto Scaling for scaling your EC2 instances, and AWS CodePipeline for continuous delivery.

ECS is a popular choice for running containerized applications in the cloud, as it provides a scalable and reliable platform for managing and running containers. You can learn more about ECS and its features in the AWS documentation.

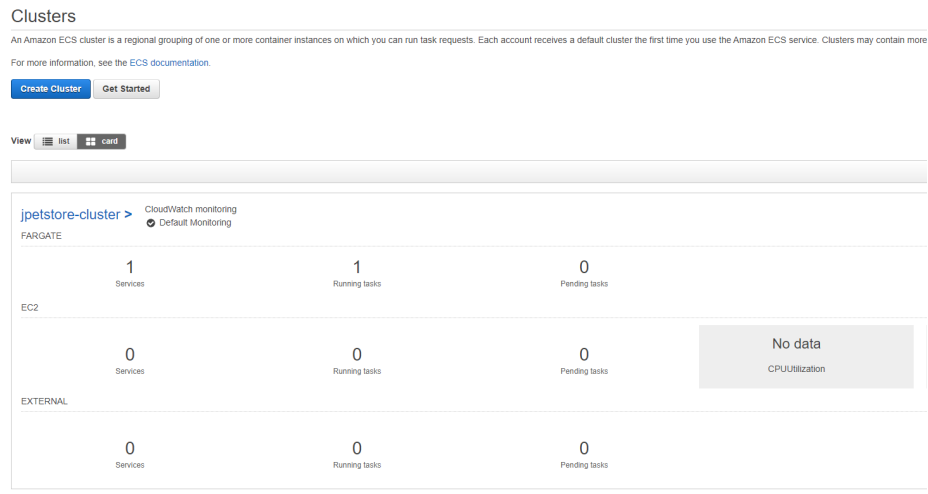


Fig. 10. AWS ECS

### 2.9.2 Integration

To integrate ECS into a GitHub Action, you can use the AWS Command Line Interface (CLI) and the `aws ecs` command in a script within the Action. Here is an example of how you can use the AWS CLI to authenticate with ECS and run a task in ECS from a GitHub Action:

First, you will need to create an IAM (Identity and Access Management) user or use an existing IAM user that has permissions to access ECS. You will need the access key ID and secret access key for this user.

In your GitHub repository, create a new Action by creating a new file in the `.github/workflows` directory. You can use the following YAML code as a starting point for your Action:

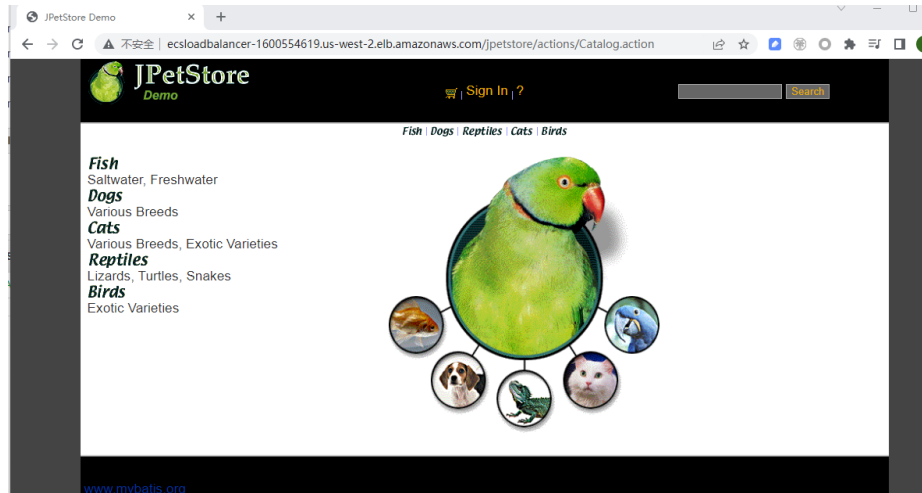


Fig. 11. Sample App

```

1  on: [push]
2
3  name: ECS
4
5  jobs:
6    deploy:
7      runs-on: ubuntu-latest
8      steps:
9        - uses: actions/checkout@v2
10       - name: Build and push Docker image
11         uses: docker/build-push-action@v1
12         with:
13           username: ${ secrets.DOCKER_USERNAME }
14           password: ${ secrets.DOCKER_PASSWORD }
15           registry: docker.io
16           repository: your-repository-name
17           tag_with_sha: true
18       - name: Deploy to ECS
19         uses: aws-actions/amazon-ecs-deploy-task-definition@v1
20         with:
21           region: us-east-1
22           cluster: your-cluster-name
23           service: your-service-name
24           task-definition: your-task-definition-name
25           image: docker.io/your-repository-name:${GITHUB_SHA}

```

```
26     force-new-deployment: true
```

```
27
```

Replace the placeholders in the YAML code with the appropriate values for your environment. In particular, you will need to provide the access key ID and secret access key for your IAM user, the region where your ECS cluster is located, and the name of your ECS cluster, task definition, and service.

Commit and push your changes to the repository. The Action will be triggered when you push to the repository, and it will run the script to authenticate with ECS and run the task in your ECS cluster.

## 2.10 Nuclei

### 2.10.1 Introduction

Nuclei GitHub Action is capable of integrating all of the Nuclei templates into continuous security workflows and making it a part of the secure software development life cycle. It is a Customizing Vulnerability Scanner that sends requests across multiple targets based on the nuclei templates leading to zero false positive or irrelevant results and provides fast scanning on various hosts.

Nuclei also automatically update the templates to the newest version and offer to scan for various protocols, including DNS, HTTP, TCP, etc. Users can find almost all kinds of security checks by performing Nuclei scans.

### 2.10.2 Integration

To deploy the tool, users must set their target application URL and store the result file in a specific folder, then input the path of the folder as indicated in the code. After the scan is complete, users may look at the report from the "Security" tab in GitHub, as mentioned above. Users can also store their log files in any designated folder, as indicated below.

```
1   nuclei-scan:
2     name: DAST Scan using Nuclei
3     needs: [zap_scan]
4     runs-on: ubuntu-latest
5     steps:
6       - uses: actions/checkout@v3
7       - name: Run ESLint
8         run: node_modules/.bin/eslint build docs lib script
9       ↪ spec-main -f
10      ↪ node_modules/@microsoft/eslint-formatter-sarif/sarif.js -o
11      ↪ results.sarif || true
12     - uses: actions/checkout@v2
13     - name: Nuclei - Vulnerability Scan
14       uses: projectdiscovery/nuclei-action@main
15     with:
```

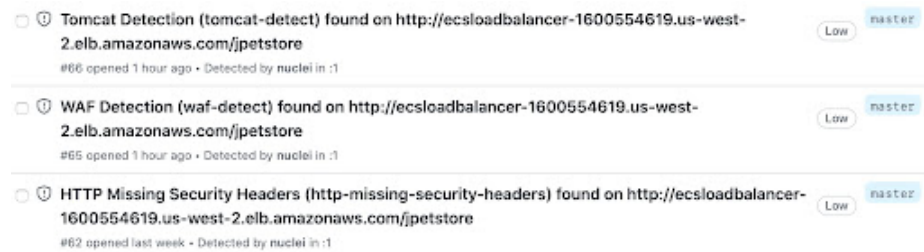


```

13         target:
14     ↪ http://ecsloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore
15         sarif-export: results.sarif
16     - name: GitHub Security Dashboard Alerts update
17       uses: github/codeql-action/upload-sarif@v2
18       with:
19         sarif_file: results.sarif
20     - name: GitHub Workflow artifacts
21       uses: actions/upload-artifact@v2
22       with:
23         name: nuclei.log
24         path: nuclei.log

```

Take one of the Nuclei scan results as an example that there are errors detected, such as "HTTP missing security headers," which could result in the application failing to prevent attackers from accessing unencrypted connections. A straightforward way to fix this is to add "X-XSS-Protection" or "X-Content-Type-Options" to the headers.



**Fig. 12.** Nuclei Scan Result

The other alert shows the application is vulnerable to exposing such applications under attack. It may make attackers aware of the version or application that is deployed.

### 2.10.3 Configuration

Users may have their template built and implemented in the scan. To deploy such a method, users can simply add their template path before the "target" key by using "templates: {PATH\_OF\_TEMPLATE}." Also, they need to replace the path of the "target" with their target path. This step sets the target of the web application that users want to scan.

## 2.11 OWASP ZAP

### 2.11.1 Introduction

OWASP ZAP (Zed Attack Proxy) is an open-source web application security

testing tool that is designed to help developers and security professionals identify vulnerabilities in web applications. One of the features of OWASP ZAP is the ability to perform a full scan of a web application, which involves a comprehensive analysis of the application's security posture.

During a full scan, OWASP ZAP will perform the following actions:

**Crawling:** OWASP ZAP will crawl the web application, analyzing the structure and content of the application and identifying potential security vulnerabilities. This may include analyzing the application's code, as well as any files, scripts, or other assets that are associated with the application.

**Active testing:** After completing the crawl, OWASP ZAP will perform active testing, in which it will attempt to exploit any vulnerabilities that it has identified by simulating attacks on the application. This may include attempting to inject malicious code, performing cross-site scripting attacks, or exploiting other types of vulnerabilities.

**Reporting:** Once the scan is complete, OWASP ZAP will generate a report detailing the results of the scan. This report will include a list of any vulnerabilities that were identified, along with information about their severity and potential impact on the application.

A full scan can be time-consuming, as it involves a thorough analysis of the entire web application. However, it can provide a comprehensive view of the application's security posture and help identify potential vulnerabilities that may not be detectable through other testing methods.

To perform a full scan with OWASP ZAP, you will need to install the tool and set up a scan configuration. This typically involves specifying the target URL for the scan, selecting the types of tests to be performed, and setting any relevant options or parameters. Once the scan is configured, you can start the scan by clicking on the "Start Scan" button in the OWASP ZAP interface. The scan will then proceed according to the configuration you have set up, and the results of the scan will be displayed in the OWASP ZAP interface once it is complete.

### 2.11.2 Integration

First, make sure you have OWASP ZAP installed on your system and a repository set up on GitHub.

Next, you'll need to create a `.github/workflows` directory in your repository, if it doesn't already exist.

Within the workflows directory, create a new file called `owasp-zap.yml` and add the following YAML code to it:

```

1 on: [push]
2
3 name: OWASP ZAP Full Scan
4
5 jobs:
6   scan:
7     runs-on: ubuntu-latest

```

```

8     steps:
9     - uses: actions/checkout@v2
10    - name: Run OWASP ZAP Full Scan
11      run: zap-full-scan.py -t http://your-target-url -r
      ↪ zap-report.html
12    env:
13      TARGET_URL: http://your-target-url
14    - name: Upload report to GitHub
15      uses: actions/upload-artifact@v2
16      with:
17        name: zap-report
18        path: zap-report.html


```

This code will trigger the action every time you push a change to your repository, and it will run a series of steps to perform a full scan of the specified target URL using OWASP ZAP. The scan results will be saved to an HTML file called `zap-report.html`, which will then be uploaded to GitHub as an artifact.

### ZAP Scan Baseline Report #16

[Open](#) [github-actions](#) bot opened this issue yesterday · 0 comments

---

 [github-actions](#) bot commented yesterday

- Site: <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com>

New Alerts

- Content Security Policy (CSP) Header Not Set [10038] total: 5:
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/robots.txt>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/sitemap.xml>
- Permissions Policy Header Not Set [10063] total: 5:
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/robots.txt>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/sitemap.xml>
- Server Leaks Version Information via "Server" HTTP Response Header Field [10036] total: 5:
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/robots.txt>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/sitemap.xml>
- Non-Storable Content [10049] total: 5:
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/jpetstore/>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/robots.txt>
  - <http://ecslloadbalancer-1600554619.us-west-2.elb.amazonaws.com/sitemap.xml>

View the following link to download the report.  
RunnerID:3721548406

**Fig. 13.** OWASP ZAP Scan Results

You should now be all set to run the action and perform a full scan with OWASP ZAP. To do this, commit and push your code to your repository. This

will trigger the action, and you should see it running in the "Actions" tab of your repository.

Once the action has been completed, you can view the results of the scan in the artifact that was uploaded to GitHub. To do this, go to the "Actions" tab of your repository and click on the action that was just run. From there, you can click on the "Artifacts" tab and download the zap-report.html file to view the scan results

### 2.11.3 Configuration

To scan your application with ZAP, all you have to do is to set the target to your application's URL or IP.

## References

1. Understanding GitHub Actions, <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>.
2. Understanding DevSecOps, [https://www.trendmicro.com/en\\_us/devops/22/a/analyzing-devsecops-vs-devops.html](https://www.trendmicro.com/en_us/devops/22/a/analyzing-devsecops-vs-devops.html).
3. SonarCloud, <https://docs.sonarcloud.io/>.
4. Amazon Elastic Container Registry Documentation, <https://docs.aws.amazon.com/ecr/index.html>.
5. Amazon Elastic Container Service Documentation, <https://docs.aws.amazon.com/ecs/index.html>.
6. OWASP ZAP Full Scan, <https://github.com/marketplace/actions/owasp-zap-full-scan>.
7. Gitleaks, <https://github.com/zricethezav/gitleaks/>.
8. Semgrep, <https://semgrep.dev/docs/category/continuous-integration-ci/>.
9. Trivy, <https://github.com/aquasecurity/trivy-action/>.
10. Snyk, <https://github.com/snyk/actions/>.
11. CodeQL, <https://github.com/github/codeql-action/>.
12. GitGuardian, <https://github.com/GitGuardian/ggshield-action/>.
13. Nuclei, <https://github.com/projectdiscovery/nuclei-action/>.