

# Sprint 3 Deliverables

## Continuous Integration with Security Testing



### 1. CI/CD Pipeline Setup with Version Control System (VCS)

- **CI/CD Tool Selection:** Explain the chosen CI/CD tool (e.g., Jenkins, GitLab CI/CD) and justify its selection based on project requirements and team familiarity.
  - Our group chose AWS tools to stage our pipeline due to the popularity and ease of access, and to gain experience and exposure to these well-used tools by many companies. As listed on our pipeline architecture, the following AWS CI/CD tools were utilized:
  - **AWS CodeBuild:** This is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. It supports multiple programming languages and can integrate with other AWS services like CodePipeline, CodeDeploy, and others.
  - **AWS CodeDeploy:** This is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, or serverless Lambda functions. It can be integrated with CodePipeline to enable automated deployments as part of the release pipeline.
  - **AWS CodePipeline:** This is a fully managed continuous delivery service that helps automate the release pipelines for applications. It integrates with other AWS services like CodeBuild, CodeDeploy, and third-party tools to enable building, testing, and deploying applications automatically
- **VCS Integration:** Describe the configuration steps to integrate the chosen CI/CD tool with a Git repository. This includes:
  - Setting up webhooks or triggers to initiate builds upon code commits- configuring a mechanism that automatically starts the CI/CD pipeline or build process whenever changes are pushed to the source code repository (e.g., Git repository).
  - Defining workflows or pipelines within the CI/CD tool to automate build stages.

#### Setting up AWS CodeBuild:

- Go to the AWS CodeBuild console or use the AWS CLI/CloudFormation to create a new CodeBuild project.
- Configure the project name, description, and other general settings.
- Set up the source code configuration

- Select the source code provider (We utilized a Vercel-based code found on GitHub)

- Provide the repository URL, branch or reference, and authentication method (e.g., AWS credentials, OAuth token). d. Configure the build environment:
- Select the operating system (e.g., Ubuntu, Amazon Linux 2), compute type, and image.
- Install any required build tools or dependencies in the build environment. e. Define the build specifications:
- Set the build commands (e.g., `npm install`, `mvn package`) to be executed during the build process.
- Configure the artifact settings, such as the artifact type (e.g., zip, S3), artifact name, and storage location (e.g., S3 bucket). f. Set up the required IAM service role with appropriate permissions for CodeBuild to access resources like S3, ECR, etc. g. Review and create the CodeBuild project.

**Code Review Enforcement:** Explain the chosen method for enforcing code reviews within the CI/CD pipeline. This could involve integrating with a code review tool (e.g., GitHub Pull Requests) or defining mandatory review stages within the pipeline itself.

To enforce code reviews within the CI/CD pipeline, we can leverage the integration with a code review tool like GitHub Pull Requests. This approach ensures that code changes are thoroughly reviewed before being merged into the main branch and deployed to production.

Here's how we can implement code review enforcement using GitHub Pull Requests:

1. **Pull Request Workflow:** Developers create a new branch for their code changes and submit a Pull Request (PR) when they want to merge their changes into the main branch.
2. **Automatic Branch Build:** The CI/CD pipeline automatically triggers a build for the branch associated with the Pull Request. This build includes running unit tests, code linting, and any other automated checks defined in the pipeline.
3. **Code Review Requirements:** We can configure GitHub to enforce specific requirements before a Pull Request can be merged, such as:
  - Minimum number of approvals from designated reviewers or team members
  - Successful completion of all required checks (e.g., build, tests, linting)
  - Resolving any merge conflicts
  - Meeting other criteria based on team policies (e.g., adhering to coding standards, documenting changes)
4. **Review Process:** Reviewers can leave comments, request changes, or approve the Pull Request after thoroughly reviewing the code changes, commit messages, and associated documentation.
5. **Merge Approval:** Once all the required reviews and checks have passed, the Pull Request can be merged into the main branch by an authorized team member or automatically merged based on the configured settings.
6. **Deployment Trigger:** Upon successful merge, the CI/CD pipeline is triggered to build and deploy the merged changes to the appropriate environment (e.g., staging, production).

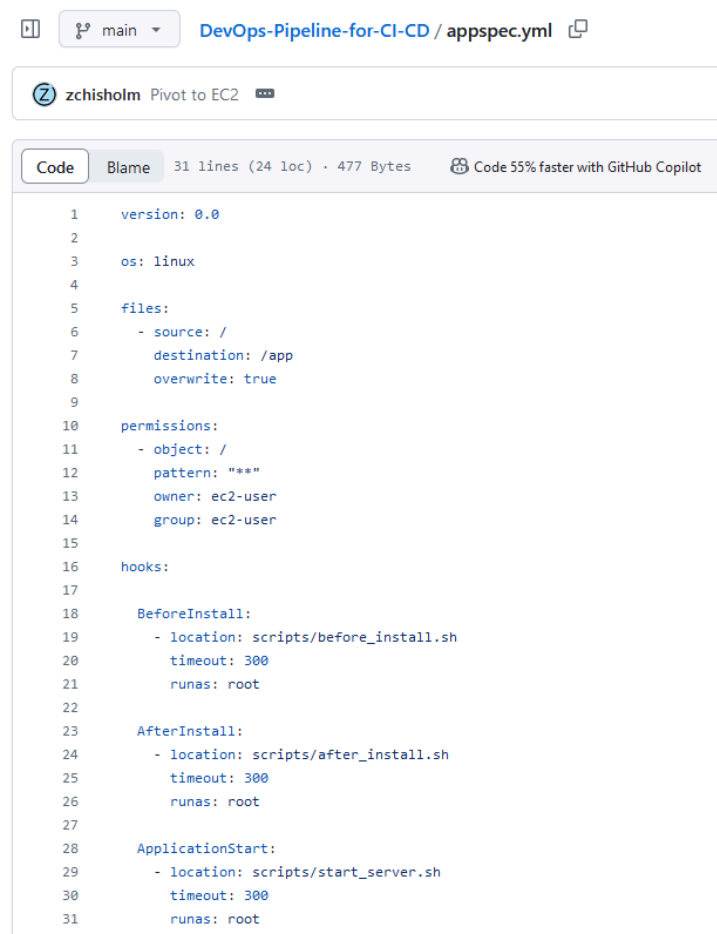
By integrating with GitHub Pull Requests, we leverage the built-in code review functionality and ensure that no code changes are merged without proper review and approval. This

approach promotes collaboration, knowledge sharing, and code quality among the development team

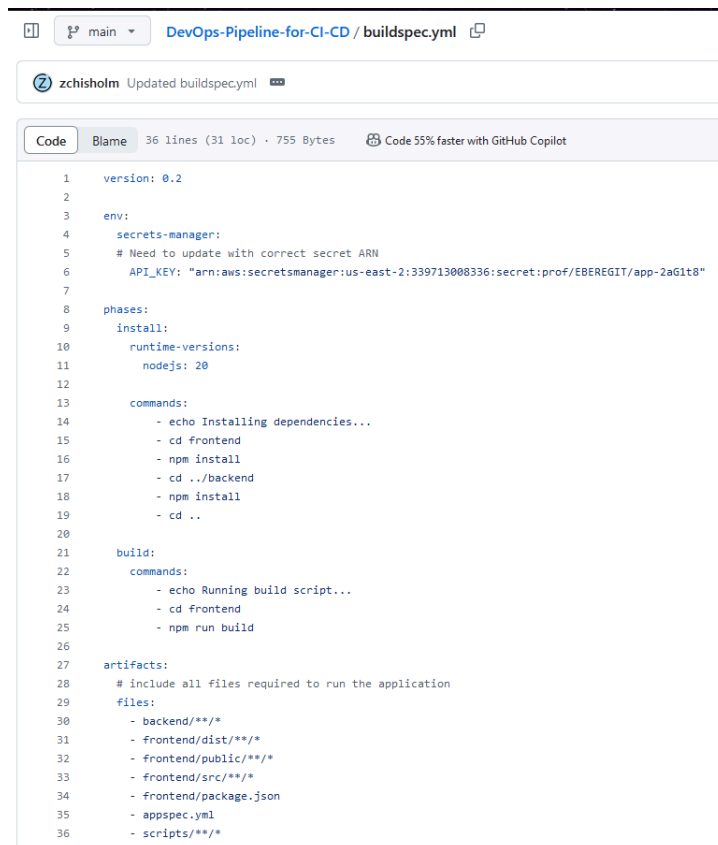
**Sample CI/CD Pipeline Configuration:** Provide code snippets or screenshots showcasing the configuration of the CI/CD pipeline, including build triggers, build stages, and code review enforcement.

- Configuration for a CI/CD pipeline using AWS CodePipeline, which is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates.

Here's a sample configuration file, written in YAML format



```
1  version: 0.0
2
3  os: linux
4
5  files:
6    - source: /
7      destination: /app
8      overwrite: true
9
10 permissions:
11   - object: /
12     pattern: "*"
13     owner: ec2-user
14     group: ec2-user
15
16 hooks:
17
18   BeforeInstall:
19     - location: scripts/before_install.sh
20       timeout: 300
21       runas: root
22
23   AfterInstall:
24     - location: scripts/after_install.sh
25       timeout: 300
26       runas: root
27
28   ApplicationStart:
29     - location: scripts/start_server.sh
30       timeout: 300
31       runas: root
```



```
1 version: 0.2
2
3 env:
4   secrets-manager:
5     # Need to update with correct secret ARN
6     API_KEY: "arn:aws:secretsmanager:us-east-2:339713008336:secret:prof/EBEREGIT/app-2aGit8"
7
8 phases:
9   install:
10    runtime-versions:
11      nodejs: 20
12
13    commands:
14      - echo Installing dependencies...
15      - cd frontend
16      - npm install
17      - cd ../backend
18      - npm install
19      - cd ..
20
21   build:
22    commands:
23      - echo Running build script...
24      - cd frontend
25      - npm run build
26
27 artifacts:
28   # include all files required to run the application
29   files:
30     - backend/**/*
31     - frontend/dist/**/*
32     - frontend/public/**/*
33     - frontend/src/**/*
34     - frontend/package.json
35     - appspec.yml
36     - scripts/**/*
```

In this configuration:

1. The Source stage is configured to trigger the pipeline when changes are pushed to the specified GitHub repository. It uses the GitHub source action type and requires GitHub authentication.
2. The Build stage runs an AWS CodeBuild project (specified by CodeBuildProject) to build and test the code. This stage takes the source code as input and produces a build artifact as output.
3. The Deploy stage runs an AWS CodeDeploy deployment to an Amazon Elastic Container Service (ECS) cluster and service. It takes the build artifact as input and deploys the application to the specified ECS service.

To enforce code reviews using GitHub Pull Requests, you can configure the GitHub repository to require a minimum number of approvals and successful checks before a Pull Request can be merged. This can be done through the GitHub repository settings or by using tools like GitHub Actions or GitHub Apps

## 2. Security Testing Integration

- **Security Testing Tool Selection:**For our application, we have chosen to incorporate both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools into our security testing strategy. This combination provides a comprehensive approach to identifying and mitigating potential vulnerabilities and security risks throughout the application's lifecycle.

- **Static Application Security Testing (SAST):** We have selected SonarQube as our SAST tool. SonarQube is an open-source platform that performs static code analysis and provides insights into code quality, security vulnerabilities, and technical debt. The choice of SonarQube is justified by the following reasons:

**1. Language Support:** SonarQube supports a wide range of programming languages, including Java, C#, Python, JavaScript, and more, which aligns with the tech stack used in our application development.

**2. Integration:** SonarQube integrates seamlessly with popular code editors, Integrated Development Environments (IDEs), and Continuous Integration/Continuous Deployment (CI/CD) tools, allowing developers to identify and fix issues early in the development cycle.

**3. Customizable Rules:** SonarQube offers a rich set of built-in security rules, and it also allows us to define custom rules tailored to our specific security requirements and coding standards.

**4. Reporting and Dashboards:** SonarQube provides detailed reports and dashboards, enabling us to track code quality, security vulnerabilities, and technical debt over time, facilitating informed decision-making and prioritization of remediation efforts.

- **Dynamic Application Security Testing (DAST):** For DAST, we have chosen OWASP ZAP (Zed Attack Proxy) as our security testing tool. OWASP ZAP is a free and open-source web application security scanner that simulates various types of attacks on running applications. We selected OWASP ZAP for the following reasons:

**1. Comprehensive Testing:** OWASP ZAP performs a wide range of security tests, including SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and other common web application vulnerabilities defined in the OWASP Top 10 and beyond.

**2. Ease of Use:** OWASP ZAP offers a user-friendly interface and automation capabilities, making it easy to integrate into our testing workflows and CI/CD pipelines.

**3. Active Community and Support:** OWASP ZAP has an active community of contributors and users, ensuring regular updates, security advisories, and ample resources for troubleshooting and knowledge sharing.

**4. Customization and Scripting:** OWASP ZAP supports scripting and customization, allowing us to extend its functionality and tailor security tests to our specific application requirements and test scenarios.

By combining SAST with SonarQube and DAST with OWASP ZAP, we can achieve a comprehensive security testing strategy that addresses both code-level vulnerabilities and runtime application security issues. SonarQube helps us catch security flaws early in

the development cycle, while OWASP ZAP tests the running application under simulated attack scenarios, providing a layered approach to security testing.

This combination of tools aligns with our application's architecture, technology stack, and security requirements, ensuring that we can identify and mitigate potential vulnerabilities effectively before deploying the application to production environment.

- **SAST Integration:** Describe how the SAST tool is integrated within the CI/CD pipeline. This includes:
  - Downloading and installing the SAST tool within the build environment.
  - Configuring the SAST tool to scan the application codebase.
  - Defining how SAST results are reported and handled within the pipeline (e.g., failing the build on critical vulnerabilities).
- **SAST- SonarQube**
  - Focuses on identifying common coding flaws, such as null pointer dereferences, buffer overflows, and resource leaks.
  - Detects potential security vulnerabilities like insecure cryptographic practices, injection flaws, and improper error handling.

**Here is a general overview of AWS CodeBuild with SonarQube:**

- **Deploy SonarQube Server**
  - Deploy the SonarQube server on an Amazon EC2 instance or using Docker containers on Amazon ECS/EKS.
- **Configure AWS CodeBuild Project**
  - Create a new AWS CodeBuild project or modify an existing one.
  - In the project settings, add a build spec file (e.g., buildspec.yml) with the following steps:
    - Install the SonarQube Scanner for your specific programming language.
    - Run the SonarQube Scanner to analyze your code and send the results to the SonarQube server.
- **Integrate with AWS CodePipeline**
  - Add the CodeBuild project as a stage in your AWS CodePipeline.
  - Configure the pipeline to trigger the CodeBuild project on code changes or specific events.
- **Review SonarQube Reports**
  - After the CodeBuild project runs, you can access the SonarQube server to review the analysis reports and identified issues.
- **Downloading and installing SAST tool/SonarQube**

In the AWS CodeBuild project configuration, you can specify the steps to download and install SonarQube or any other SAST tool during the build process. Here's an example for SonarQube:

a. Add commands to download the SonarQube Scanner (a tool that analyzes the code and sends the results to the SonarQube server) in the build environment:

**# Download the SonarQube Scanner**

```
wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.7.0.2747-linux.zip
```

```
unzip sonar-scanner-cli-4.7.0.2747-linux.zip
```

```
# Download the SonarQube Scanner
wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.7.0.2747-linux.zip
unzip sonar-scanner-cli-4.7.0.2747-linux.zip
```

b. Set up the SonarQube server URL and authentication token as environment variables in the CodeBuild project:

**# Set SonarQube server and authentication details export**

```
SONAR_HOST_URL=http://sonarqube.example.com export SONAR_TOKEN=abc123def456
```

```
# Set SonarQube server and authentication details
export SONAR_HOST_URL=http://sonarqube.example.com
export SONAR_TOKEN=abc123def456
```

## 2. Configuring SAST Tool (SonarQube) to Scan the Application Codebase:

After installing the SonarQube Scanner, you can configure it to scan your application's codebase as part of the CodeBuild build commands:

```
# Run the SonarQube Scanner ./sonar-scanner-4.7.0.2747-linux/bin/sonar-scanner \ -
Dsonar.projectKey=my-app \ -Dsonar.sources=./src \ -
Dsonar.host.url=$SONAR_HOST_URL \ -Dsonar.login=$SONAR_TOKEN
```



```
# Run the SonarQube Scanner
./sonar-scanner-4.7.0.2747-linux/bin/sonar-scanner \
  -Dsonar.projectKey=my-app \
  -Dsonar.sources=./src \
  -Dsonar.host.url=$SONAR_HOST_URL \
  -Dsonar.login=$SONAR_TOKEN
```

This command will analyze the code in the `./src` directory and send the results to the SonarQube server specified by `$SONAR_HOST_URL`, using the provided authentication token.

### 3. Defining How SAST Results are Reported and Handled within the Pipeline:

After the SonarQube Scanner has analyzed the code, you can define how the SAST results should be reported and handled within the CI/CD pipeline:

#### a. Reporting SAST Results:

- SonarQube provides a web interface where you can view the analysis results, including code quality metrics, vulnerabilities, and other issues.
- You can integrate SonarQube with tools like Slack, Microsoft Teams, or email to receive notifications about the analysis results.

#### b. Handling SAST Results within the Pipeline:

- You can configure the CodeBuild project to fail the build if the SAST tool (e.g., SonarQube) reports critical vulnerabilities or quality gate failures.
- This can be achieved by adding a command to check the SonarQube quality gate status after the analysis and failing the build if the status is not successful.

```
# Check SonarQube quality gate status
```

```
STATUS=$(curl -u $SONAR_TOKEN: "$SONAR_HOST_URL/api/qualitygates/project_status?projectKey=my-app" | jq -r '.projectStatus.status')
```

```
# Fail the build if quality gate is not successful
```

```
if [ "$STATUS" != "OK" ]; then
```

```
  echo "SonarQube quality gate failed: $STATUS"
```



```
exit 1
```

```
fi
```

```
# Check SonarQube quality gate status
STATUS=$(curl -u $SONAR_TOKEN: "$SONAR_HOST_URL/api/qualitygates/project_status?projectKey=$PROJECT_KEY")

# Fail the build if quality gate is not successful
if [ "$STATUS" != "OK" ]; then
    echo "SonarQube quality gate failed: $STATUS"
    exit 1
fi
```

- Alternatively, you can add an approval step in AWS CodePipeline after the SAST analysis stage, where human intervention is required to review and approve the SAST results before proceeding with the deployment.

By integrating SAST tools like SonarQube into the CI/CD pipeline, you can catch security vulnerabilities and code quality issues early in the development process, ensuring that only secure and high-quality code is deployed to production environments.

- **DAST Integration (Optional):** If applicable, describe the integration of a DAST tool within the CI/CD pipeline. This might involve setting up a test environment and configuring the DAST tool to scan the deployed application.

Integrating a DAST (Dynamic Application Security Testing) tool like OWASP ZAP within the CI/CD pipeline involves setting up a test environment and configuring the tool to scan the deployed application. Here's how you can integrate OWASP ZAP with your AWS CI/CD pipeline:

1. **Setting up a Test Environment:**
  - Create a separate test environment for running DAST scans. This could be a temporary EC2 instance, an ECS cluster, or a Lambda function, depending on your application's architecture.
  - Deploy your application to this test environment during the CI/CD pipeline's deployment stage.
2. **Downloading and Installing OWASP ZAP:**
  - In your CodeBuild project, add commands to download and install OWASP ZAP as part of the build process.
  - You can download the ZAP binary or use a Docker image for easier installation and configuration.
3. **Configuring OWASP ZAP:**
  - After installing ZAP, configure it to scan your deployed application in the test environment.

- Set the target URL, scan policies, and any authentication credentials required for the scan.
- You can automate this process by creating a ZAP script or using the ZAP API to programmatically configure and run the scan.

#### 4. **Running the DAST Scan:**

- Add commands to your CodeBuild project's buildspec.yml file to execute the OWASP ZAP scan against the deployed application in the test environment.
- This can be done by running the ZAP script or invoking the ZAP API with the appropriate configurations.

#### 5. **Handling DAST Scan Results:**

- OWASP ZAP generates reports containing information about vulnerabilities and other findings from the scan.
- You can configure CodeBuild to capture and store the ZAP report as an artifact for further analysis or reporting.
- Alternatively, you can integrate ZAP with other reporting tools like SonarQube or GitHub Security Advisories to centralize the security findings.

#### 6. **Failing the Build or Deployment:**

- Based on the severity of the vulnerabilities found during the DAST scan, you can configure CodeBuild to fail the build or halt the pipeline if critical vulnerabilities are detected.
- This can be achieved by parsing the ZAP report and checking for specific vulnerability types or severity levels.
- You can also implement an approval step in CodePipeline, where human intervention is required to review and approve the DAST scan results before proceeding with the deployment to production.

#### 7. **Cleanup and Tear Down:**

- After the DAST scan is completed and the results are analyzed, ensure that you have steps in your pipeline to tear down the temporary test environment used for the scan.
- This can include terminating EC2 instances, deleting ECS resources, or removing Lambda functions and related resources.

By integrating DAST tools like OWASP ZAP into your CI/CD pipeline, you can identify and address security vulnerabilities in your application's runtime behavior before deploying it to production environments. This helps to improve the overall security posture of your application and reduce the risk of security breaches.

- **Security Testing Results Reporting:** Explain how security testing results are reported and visualized for developers (e.g., integrated reports within the CI/CD interface, email notifications).

In an AWS CI/CD pipeline, there are several ways to report and visualize security testing results for developers. Here are some common approaches:

#### 1. **Integrated Reports within the CI/CD Interface:**

- **AWS CodePipeline:**
  - CodePipeline integrates with other AWS services like CodeBuild and CodeDeploy, which can be used to run security tests and generate reports.

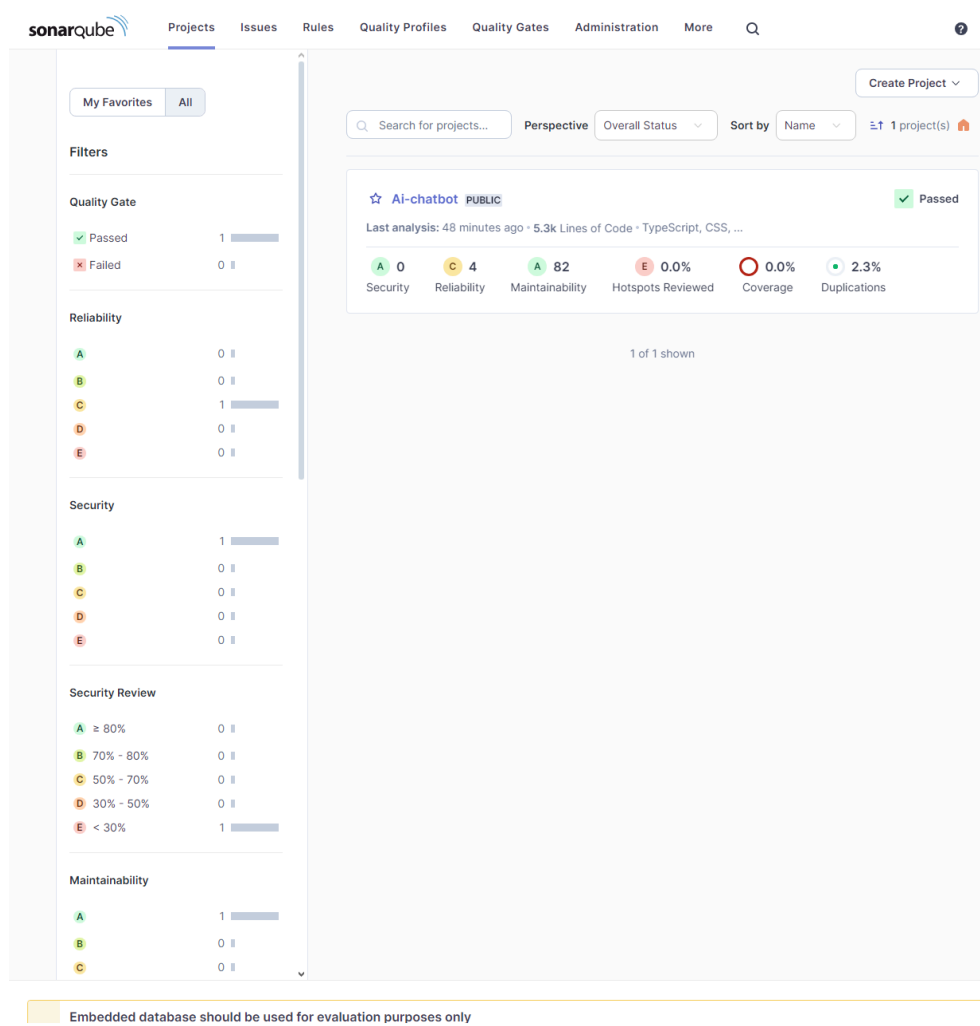
- The security testing results (e.g., SAST, DAST, or other custom reports) can be stored as artifacts in an S3 bucket or other storage locations.
- Within the CodePipeline console, you can view and access these artifact reports for each pipeline execution.
- **AWS CodeBuild:**
  - In CodeBuild, you can configure the build project to generate and store security testing reports as build artifacts.
  - These reports can be accessed through the CodeBuild console or downloaded from the artifact location (e.g., S3 bucket).
  - Additionally, you can configure CodeBuild to publish the reports to AWS CodePipeline as artifacts for easier access within the pipeline interface.

## 2. Reporting Tools Integration:

- Integrate security testing tools like SonarQube, OWASP ZAP, or other commercial tools with your CI/CD pipeline.
- These tools often provide web interfaces or dashboards to visualize and analyze the security testing results.
- Configure the tools to publish the reports as artifacts or integrate them with the CI/CD pipeline for easier access.
- **Email Notifications:**
  - Set up Amazon Simple Email Service (SES) or Simple Notification Service (SNS) to send email notifications about security testing results.
  - In your CodeBuild or CodePipeline projects, add commands to trigger email notifications based on the security testing results.
  - You can include the summary of vulnerabilities, severity levels, and attach the detailed reports as email attachments.
  - Configure email distribution lists or individual recipients to receive these notifications.
- **AWS CloudWatch Logs and Metrics:**
  - Capture security testing logs and metrics in AWS CloudWatch Logs.
  - Create CloudWatch Logs Metric Filters to extract relevant security metrics from the logs.
  - Set up CloudWatch Alarms to monitor these metrics and trigger notifications or actions based on predefined thresholds or conditions.
  - Visualize the security testing metrics using CloudWatch Dashboards or integrate with other monitoring tools like Amazon EventBridge, AWS Lambda, or third-party tools.
- **Third-Party Reporting and Visualization Tools:**
  - Integrate with third-party reporting and visualization tools like Datadog, New Relic, or Splunk.
  - Configure these tools to ingest security testing logs and reports from your CI/CD pipeline.
  - Leverage their dashboards, alerts, and visualization capabilities to monitor and analyze the security testing results.
- **Slack or Microsoft Teams Notifications:**
  - Set up integrations with collaboration tools like Slack or Microsoft Teams to receive notifications about security testing results.
  - Configure webhooks or APIs to post security testing summaries, vulnerabilities, or detailed reports as messages in designated channels.

- Leverage the rich formatting and attachment capabilities of these tools to present the security testing results in an organized and readable manner.

● **Sample CI/CD Integration with Security Testing Tools:** Provide code snippets or screenshots showcasing the integration of security testing tools within the pipeline configuration.



sonarqube

ProjectsIssuesRulesQuality ProfilesQuality GatesAdministrationMore

My IssuesAll

Filters

Clear All Filters

Clean Code Attribute

Consistency1Intentionality3Adaptability0Responsibility0

Software Quality1x

Security0Reliability4Maintainability82

Add to selectionCtrl + click

> Severity?

> Type

> Scope

> Status

> Security Category

> Creation Date

> Language

> Rule

> Tan

Bulk Change

Select issues

Navigate to issue

4 issues20min effort

AI-chatbot / components/chat-panel.tsx

Visible, non-interactive elements with click handlers must have at least one keyboard listener.

Intentionality

Reliability

accessibilityreact

Open

Not assigned

L71 · 5min effort · 50 minutes ago · Bug · Minor

Avoid non-native interactive elements. If using native HTML is not possible, add an appropriate role and support for tabbing, mouse, keyboard, and touch inputs to an interactive content element.

Consistency

Reliability

Maintainability

accessibilityreact

Open

Not assigned

L71 · 5min effort · 50 minutes ago · Code Smell · Minor

Unexpected constant truthiness on the left-hand side of a '||' expression.

Intentionality

Reliability

No tags

Open

Not assigned

L71 · 5min effort · 50 minutes ago · Bug · Major

Add a "yield" statement to this generator.

Intentionality

Reliability

api-designes2015

Open

Not assigned

L296 · 5min effort · 50 minutes ago · Bug · Major

4 of 4 shown

Embedded database should be used for evaluation purposes only