
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

SEC314

Automate security analysis and code reviews with machine learning

Jas Chhabra (he/him)

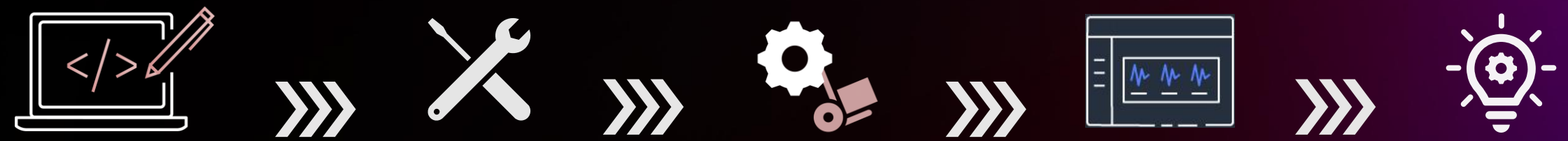
Sr. Software Dev Manager, AI DevOps
Amazon Web Services

Leo da Silva (he/him)

Security Solutions Architect
Amazon Web Services

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

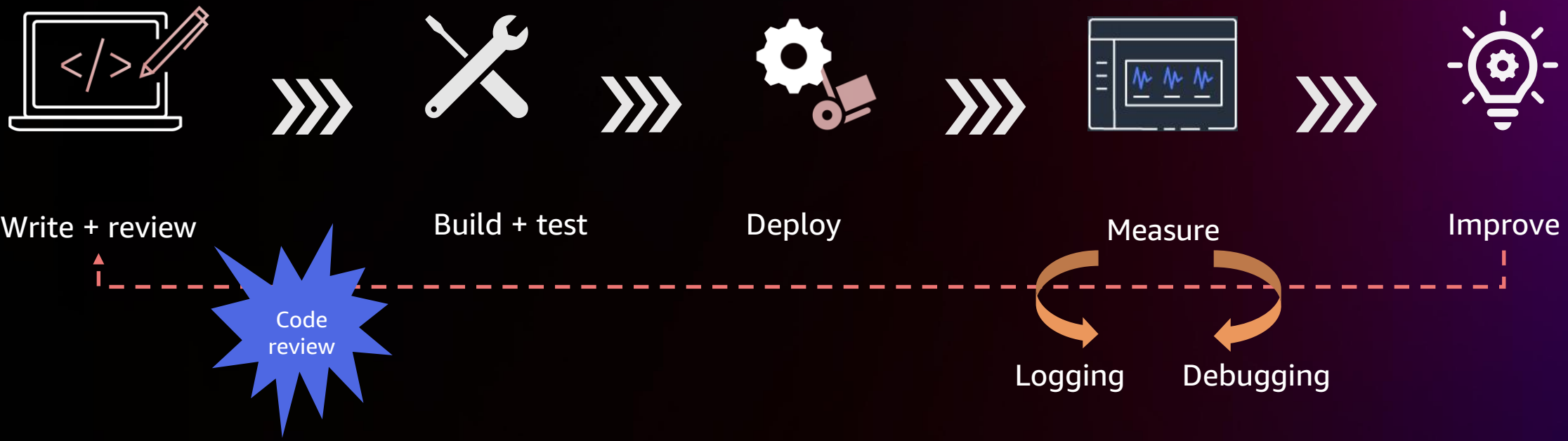
A typical software workflow



Write + review → Build + test → Deploy → Measure → Improve

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A typical software workflow

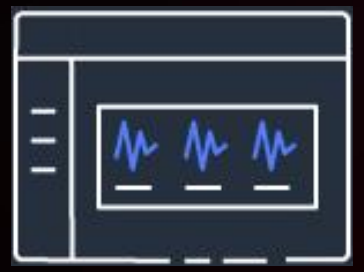


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Challenges in the application lifecycle



It is time-consuming for developers to identify problematic code



Code analysis tools lack industry-standard best practices for code security



Code security processes lie outside of the developer workflow, require rework and context switching

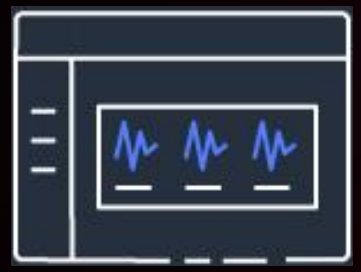
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Challenges in the application lifecycle



It is time-consuming for developers to identify problematic code



Code analysis tools lack industry-standard best practices for code security



Code security processes lie outside of the developer workflow, require rework and context switching

Challenge for security engineers

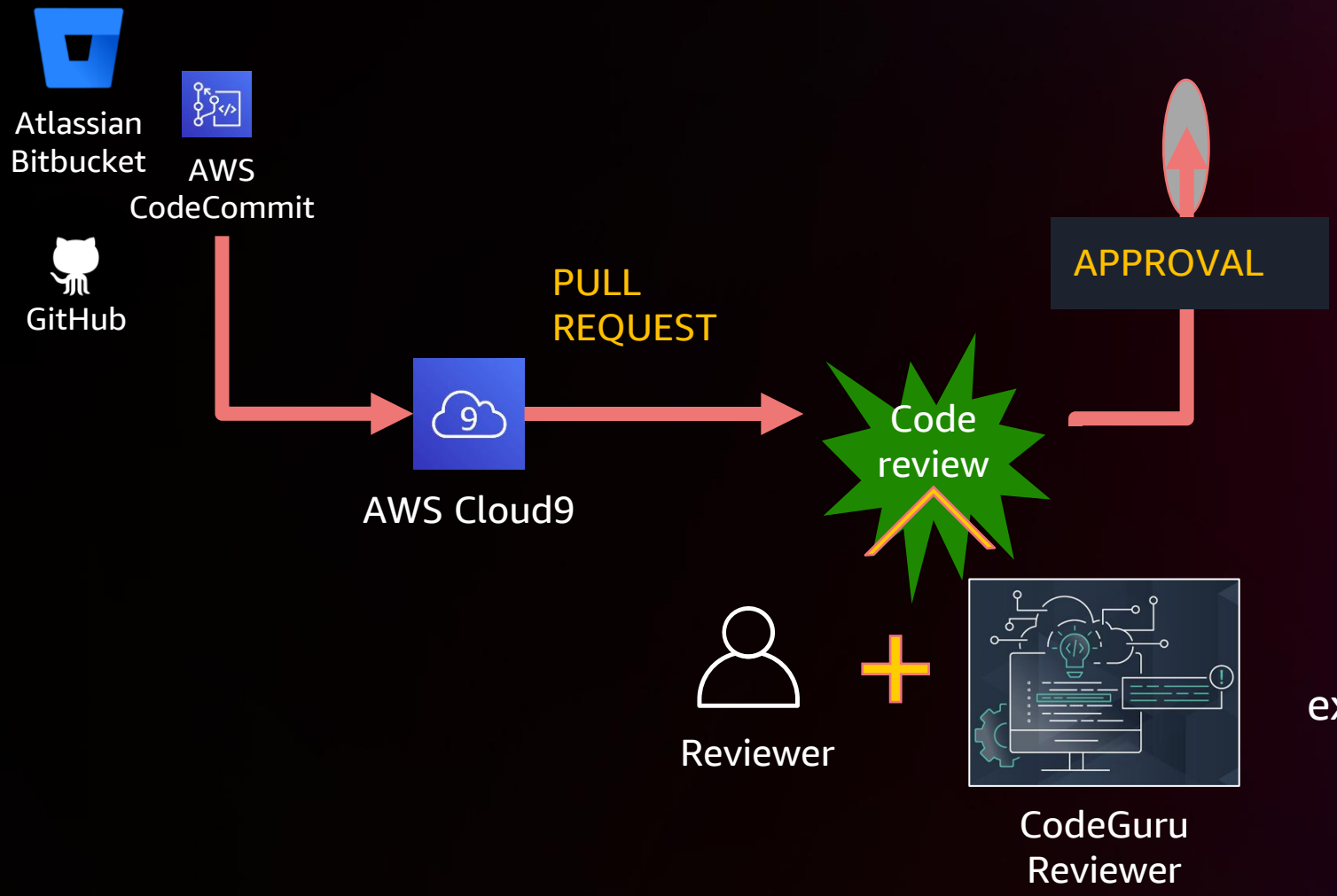
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is Amazon CodeGuru Reviewer?

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

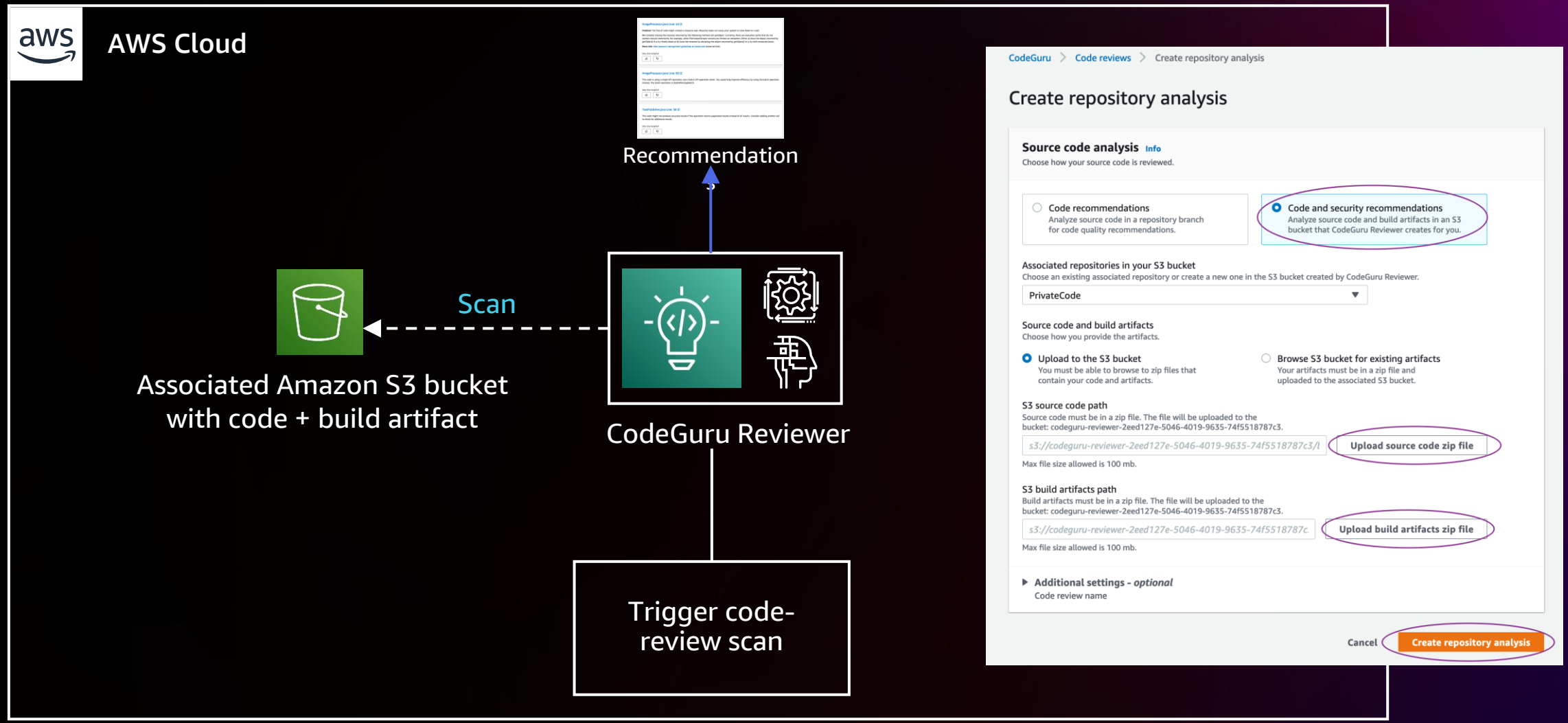
1. Code review – Pull request



Code review buddy with expertise in security, AWS best practices, resource leaks…

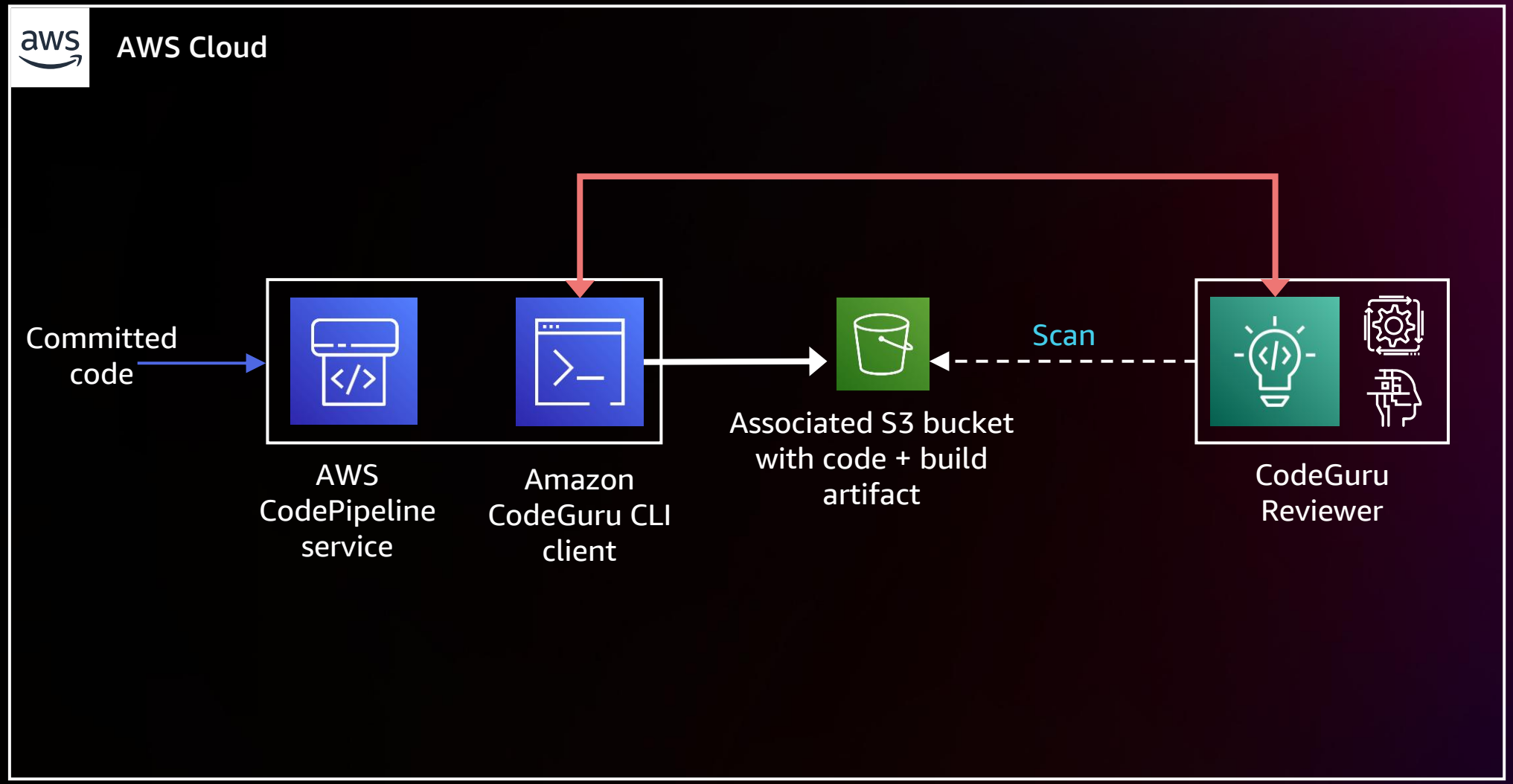
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

2. Code review – Dashboard



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

3. Using CodeGuru CLI from the CI pipeline



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security categories

- CodeGuru Reviewer supports Java and Python with 96+ unique security rules
- AWS API security best practices
- Java Crypto library best practices

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security categories

- Secure web applications
- Sensitive information leak
- AWS security best practices

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security detectors

- OWASP Top 10
- CWE Top 25
- Access-control
- Availability
- Batch-operations
- Concurrency
- Configuration
- Consistency
- Deserialization
- Information-leak
- Maintainability
- Resource-leak
- Secrets
- Security Misconfiguration
- Sensitive-data-exposure
- Cookies
- Cryptography
- Data-integrity
- Injection
- Ldap
- Log4j
- Logging
- Networking
- Null-check
- Python-annotations
- Race-condition
- Sqli
- Xml
- Amazon-dynamodb
- Amazon-ec2
- Amazon-s3
- Amazon-sns
- Android
- AWS-cloudformation
- AWS-iam
- AWS-javascript-sdk
- AWS-java-sdk
- AWS-kms
- AWS-lambda
- AWS-python-sdk
- AWS-sdk

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security detectors – Log injection

Log injection High

User-provided inputs must be sanitized before they are logged. An attacker can use unsanitized input to break a log's integrity, forge log entries, or bypass log monitors.

Detector ID
java/log-injection@v1.0

Category
Security

Common Weakness Enumeration (CWE)
CWE-117

Tags
data-integrity # injection # log4j # owasp-top10

Noncompliant example

```
1  @RequestMapping("/example.htm")
2  public ModelAndView loggingNonCompliant(HttpServletRequest request) {
3      ModelAndView result = new ModelAndView("success");
4      String userId = request.getParameter("userId");
5      result.addObject("userId", userId);
6      // More logic here to populate the result.
7      // Noncompliant: unsanitized input is logged.
8      log.info("Successfully processed request for user ID: {}.", userId);
9      return result;
10 }
```

Compliant example

```
1  @RequestMapping("/example.htm")
2  public ModelAndView loggingCompliant(HttpServletRequest request) {
3      ModelAndView result = new ModelAndView("success");
4      String userId = request.getParameter("userId");
5      String sanitizedUserId = sanitize(userId);
6      result.addObject("userId", sanitizedUserId);
7      // More logic here to populate the result.
8      // Compliant: input is sanitized before logging.
9      log.info("Successfully processed request for user ID: {}.", sanitizedUserId);
10     return result;
11 }
12 private static String sanitize(String userId) {
13     // Remove all chars except digits.
14     return userId.replaceAll("\\D", "");
15 }
```

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security detectors – AWS security best practices

Client-side KMS reencryption

High

Client-side decryption followed by reencryption is inefficient and can lead to sensitive data leaks. The `reencrypt` APIs allow decryption followed by reencryption on the server side. This is more efficient and secure.

Detector ID
java/aws-kms-reencryption@v1.0

Category
Security

Common Weakness Enumeration (CWE)
CWE-310 , CWE-311

Tags
aws-java-sdk # aws-kms # efficiency # information-leak # owasp-top10

Compliant example

```
1  public void withReEncryptCompliant(KmsClient client, SdkBytes encryptedContent) {
2      // Compliant: uses a ReEncryptRequest which runs server-side.
3      ReEncryptRequest req = ReEncryptRequest.builder()
4              .ciphertextBlob(encryptedContent)
5              .destinationKeyId("my-key-id")
6              .build();
7      client.reEncrypt(req).ciphertextBlob();
8  }
```

Noncompliant example

```
1  public void withoutReEncryptNoncompliant(KmsClient client, SdkBytes encryptedContent) {
2      DecryptRequest decryptRequest = DecryptRequest.builder()
3              .ciphertextBlob(encryptedContent)
4              .build();
5      SdkBytes plaintext = client.decrypt(decryptRequest).plaintext();
6      // Noncompliant: client-side decrypt immediately followed by encrypt.
7      EncryptRequest encryptRequest = EncryptRequest.builder()
8              .keyId("my-key-id")
9              .plaintext(plaintext)
10             .build();
11     client.encrypt(encryptRequest);
12 }
```

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security detectors – Cryptography

Insecure cryptography

Critical

Misuse of cryptography-related APIs can create security vulnerabilities. This includes one or more of the following: algorithms with known weaknesses, certain padding modes, lack of integrity checks, and insufficiently large key sizes.

Detector ID
java/insecure-cryptography@v1.0

Category
Security

Common Weakness Enumeration (CWE)
CWE-327 , CWE-328

Tags
cryptography # owasp-top10

Noncompliant example

```
1 public void keyPairGeneratorNoncompliant() throws NoSuchAlgorithmException {
2     KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
3     // Noncompliant: keysize too small for this algorithm.
4     keyPairGenerator.initialize(128);
5     keyPairGenerator.genKeyPair();
6 }
```

Compliant example

```
1 public void keyPairGeneratorCompliant() throws NoSuchAlgorithmException {
2     KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
3     // Compliant: keysize sufficient for this algorithm.
4     keyPairGenerator.initialize(4096);
5     keyPairGenerator.genKeyPair();
6 }
```

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security detectors – Cross-site scripting

Cross-site scripting High

User-controllable input must be sanitized before it's included in output used to dynamically generate a web page. Unsanitized user input can introduce cross-side scripting (XSS) vulnerabilities that can lead to inadvertedly running malicious code in a trusted context.

Detector ID
java/cross-site-scripting@v1.0

Category
Security

Common Weakness Enumeration (CWE)
CWE-79

Tags
injection # owasp-top10 # top25-cwes

Noncompliant example

```
1  public ModelAndView inputSanitizationNonCompliant(@RequestParam String favoriteColor) {
2      ModelAndView modelAndView = new ModelAndView();
3      modelAndView.setViewName("jsp/example.jsp");
4      // Noncompliant: user-supplied parameter might contain malicious content.
5      modelAndView.addObject("preferredColor", favoriteColor);
6      return modelAndView;
7  }
```

Compliant example

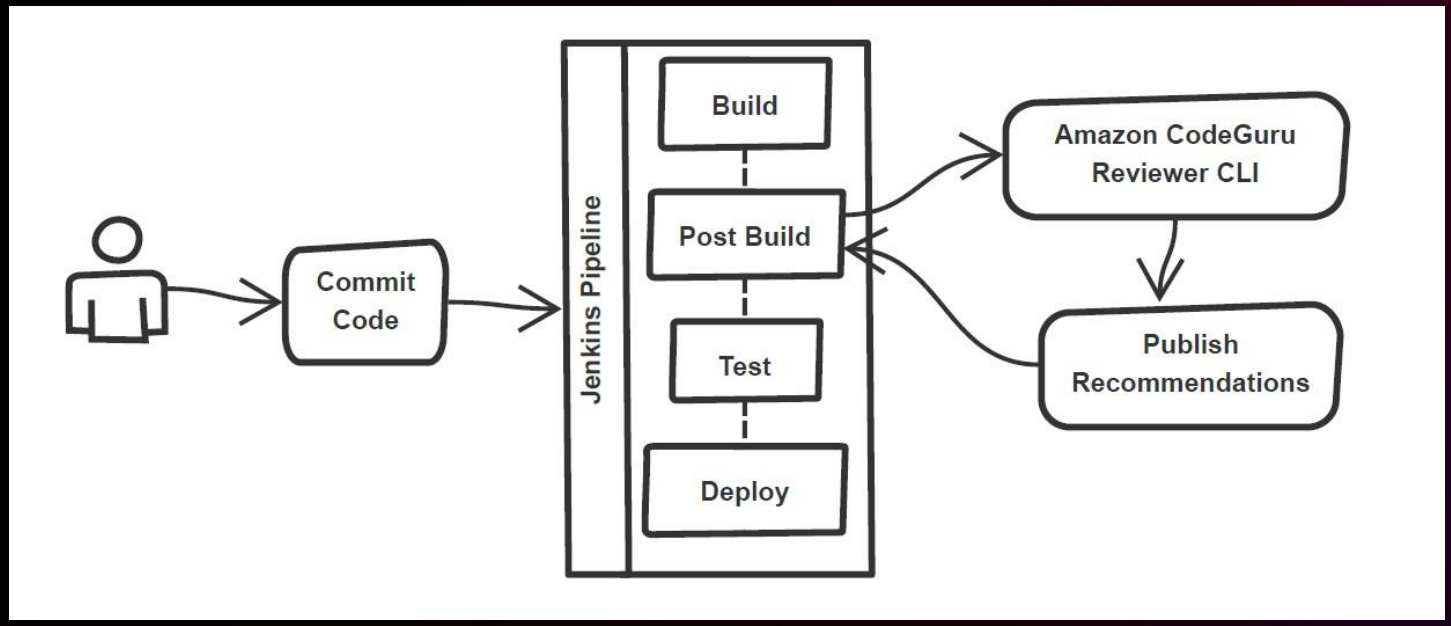
```
1  public ModelAndView inputSanitizationCompliant(@RequestParam String favoriteColor) {
2      ModelAndView modelAndView = new ModelAndView();
3      modelAndView.setViewName("jsp/example.jsp");
4      // Compliant: user-supplied parameter must be in allow-list.
5      if (favoriteColor.matches("[a-z]+")) {
6          modelAndView.addObject("preferredColor", favoriteColor);
7      } else {
8          throw new IllegalArgumentException("Invalid color!");
9      }
10     return modelAndView;
11 }
```

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Case study – CI/CD security scans with the CodeGuru CLI

“During one of our technical retrospectives, we noticed the need to integrate Amazon CodeGuru recommendations in our build pipelines hosted on Jenkins. Not all our developers can run or check CodeGuru recommendations through the AWS console. Incorporating CodeGuru CLI in our build pipelines acts as an important quality gate and ensures that our developers can immediately fix critical issues.”

-Lead DevOps at Mastercard



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CodeGuru Reviewer CLI features

Universal
Works in any CI/CD pipeline





Automatic
Block pipelines if any critical vulnerabilities are found

Easy setup
Only one command line required

```bash
aws-codeguru-cli --root-dir ./ --build target/classes --src src --output ./output
```

Integrate
Receive findings in JSON, HTML, SARIF formats

Download | More Info

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CodeGuru Reviewer CLI CI/CD workflow

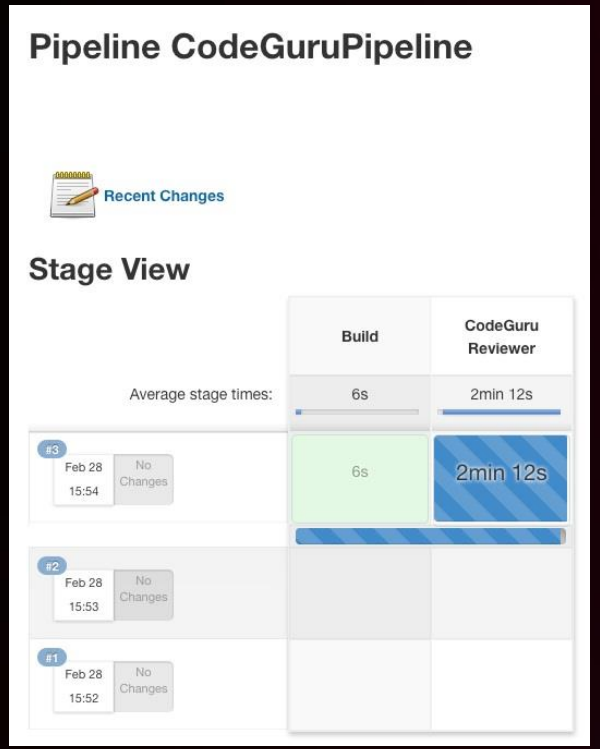
1. Customer adds CodeGuru stage into existing CI/CD pipeline

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                // Get code from a GitHub repository
                git clone https://github.com/aws-samples/amazon-codeguru-reviewer-java-detectors.g

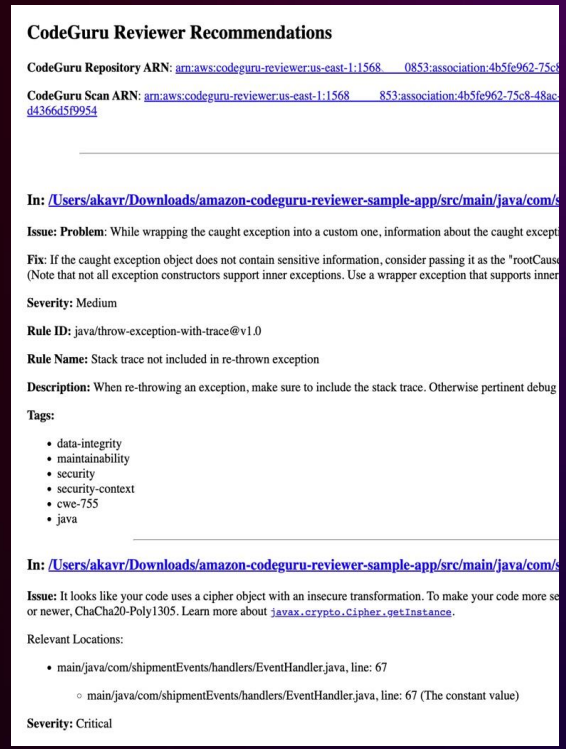
                // Run Maven on a Unix agent
                sh "mvn clean compile"

                // To run Maven on a Windows agent, use following
                // bat "mvn -Dmaven.test.failure.ignore=true clean package"
            }
        }
        stage('CodeGuru Reviewer') {
            steps{
                sh 'ls -lsa *'
                sh 'pwd'
                // Here we're setting an absolute path, but we can
                // also use JENKINS environment variables
                sh '''
                    export BASE=/var/jenkins_home/workspace/CodeGuruPipeline/amazon-codeguru-revie
                    export SRC=${BASE}/src
                    export OUTPUT = ./output
                    /home/codeguru/aws-codeguru-cli/bin/aws-codeguru-cli --root-dir $BASE --build
                    '''
            }
        }
        stage('Checking findings'){
            steps{
                // In this example we are stopping our pipline on
                // detecting Critical findings. We are using jq
                // to count occurrences of Critical severity
                sh '''
                CNT = $(cat ./output/recommendations.json |jq '.[] | select(.severity--"Critical")
                if (( $CNT > 0 )); then
                  echo "Critical findings discovered. Failing."
                  exit 1
                fi
                '''
            }
        }
    }
}
```

2. Customer triggers and tracks pipeline scan after each commit



3. Customer reviews detailed results



CodeGuru Reviewer Recommendations

CodeGuru Repository ARN: arn:aws:codeguru-reviewer:us-east-1:1568 0853:association:4b5fe962-75c8

CodeGuru Scan ARN: arn:aws:codeguru-reviewer:us-east-1:1568 853:association:4b5fe962-75c8-48ac-d4366d5f9954

In: /Users/akavr/Downloads/amazon-codeguru-reviewer-sample-app/src/main/java/com/s

Issue: Problem: While wrapping the caught exception into a custom one, information about the caught except

Fix: If the caught exception object does not contain sensitive information, consider passing it as the "rootCause (Note that not all exception constructors support inner exceptions. Use a wrapper exception that supports inner

Severity: Medium

Rule ID: java/throw-exception-with-trace@v1.0

Rule Name: Stack trace not included in re-thrown exception

Description: When re-throwing an exception, make sure to include the stack trace. Otherwise pertinent debug

Tags:

- data-integrity
- maintainability
- security
- security-context
- cwe-755
- java

In: /Users/akavr/Downloads/amazon-codeguru-reviewer-sample-app/src/main/java/com/s

Issue: It looks like your code uses a cipher object with an insecure transformation. To make your code more se or newer, ChaCha20-Poly1305. Learn more about `javax.crypto.Cipher.getInstance`.

Relevant Locations:

- main/java/com/shipmentEvents/handlers/EventHandler.java, line: 67
  - main/java/com/shipmentEvents/handlers/EventHandler.java, line: 67 (The constant value)

Severity: Critical

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Summary

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Other relevant re:Invent sessions

SEC307: Ship securely: Automated security testing for developers

SEC322: Transform builder velocity with security

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!



Please complete the session survey in the **mobile app**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.