PRJ-DOP-020: Secure DevSecOps CI/CD Pipeline

Certification: AWS Certified DevOps Engineer – Professional
Domain: SDLC Automation & Security

---

1. Project Overview

This project demonstrates how to build a secure CI/CD pipeline by integrating security practices directly into the DevOps workflow, a practice known as **DevSecOps**. The goal is to "shift left," meaning security checks are performed early and often in the development lifecycle, rather than being an afterthought. This approach helps to identify and remediate vulnerabilities before they reach production, reducing risk and cost.

We will enhance a standard CI/CD pipeline (similar to PRJ-SAP-018) by embedding multiple layers of automated security testing. This includes **Static Application Security Testing (SAST)** with Amazon CodeGuru, **Software Composition Analysis (SCA)** for third-party dependencies, and **Dynamic Application Security Testing (DAST)** against a deployed staging environment. All findings will be aggregated into **AWS Security Hub** for centralized visibility.

Key Objectives

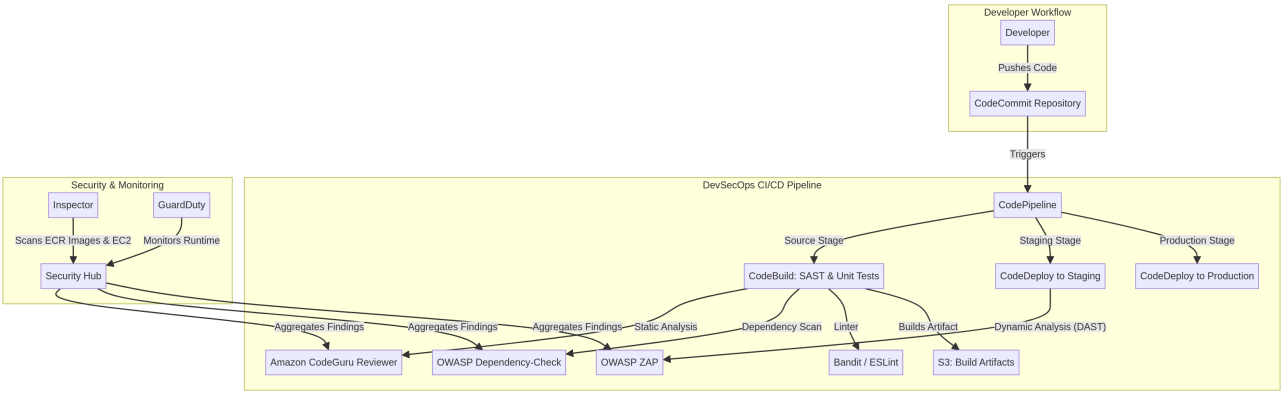
- Integrate automated security analysis into an AWS CodePipeline workflow.
- Use **Amazon CodeGuru Reviewer** to perform static code analysis and identify potential bugs and security flaws.
- Implement open-source tools like **OWASP Dependency-Check** to scan for vulnerabilities in third-party libraries.
- Use **Amazon Inspector** to scan container images in ECR for known CVEs.

- Perform dynamic application security testing (DAST) on a running application in a staging environment.
- Aggregate findings from all security tools into AWS Security Hub to provide a single pane of glass for vulnerabilities.

---

2. Architecture

The architecture embeds security scanning and testing stages throughout the CI/CD pipeline.



DevSecOps Pipeline Stages:

1. **Source Stage:** A developer commits code to **CodeCommit**. This automatically triggers the **CodePipeline**.
2. **Build & Static Analysis Stage:** This stage is handled by a single **CodeBuild** project with multiple phases:
    - **SAST (Static Application Security Testing):**
        - **Amazon CodeGuru Reviewer** is associated with the repository and automatically analyzes pull requests and full repository scans, providing recommendations on code quality and security.
        - Linters like **Bandit** (for Python) or **ESLint** (for JavaScript) are run to enforce code quality standards.
    - **SCA (Software Composition Analysis):**
        - **OWASP Dependency-Check** or a similar tool is run to scan the project's dependencies (e.g., `package.json`, `requirements.txt`)

against a database of known vulnerabilities (CVEs).

- **Unit Tests:** Standard unit tests are run to validate code functionality.
- **Build:** If all previous steps pass, the application artifact (e.g., a Docker image) is built.
- **Image Scan:** The newly built Docker image is pushed to **Amazon ECR**, which is configured to automatically trigger an **Amazon Inspector** scan to check for OS-level and package vulnerabilities within the container image.

3. **Deploy to Staging Stage:**
   - If the build and scan stage is successful, **CodeDeploy** deploys the application to a staging environment.
4. **Dynamic Analysis Stage:**
   - **DAST (Dynamic Application Security Testing):** Another CodeBuild project is triggered, which runs a DAST tool like **OWASP ZAP (Zed Attack Proxy)** against the live staging endpoint. ZAP actively probes the running application for vulnerabilities like SQL injection, Cross-Site Scripting (XSS), etc.
5. **Production Deployment:**
   - After passing all tests and an optional manual approval, the pipeline deploys the application to production.
6. **Centralized Findings:**
   - Throughout the process, findings from CodeGuru, Inspector, GuardDuty, and other integrated tools are aggregated in **AWS Security Hub**, providing a centralized dashboard for the security team to monitor the organization's security posture.

---

3. Prerequisites

- An AWS account with administrative permissions.
- A CI/CD pipeline for a containerized application (you can adapt the one from PRJ-SAP-018).

- A sample application with some known (or simulated) vulnerabilities for testing.

---

4. Step-by-Step Implementation Guide

We will focus on modifying a CodeBuild `buildspec.yml` file to include SAST and SCA steps.

Step 4.1: Integrate Amazon CodeGuru Reviewer (SAST)

1. Go to the **Amazon CodeGuru console**.
2. Go to **Code reviews -> Associate repository**.
3. Choose your **CodeCommit** repository.
4. This will associate CodeGuru with your repository. It will now automatically analyze code on pull requests. You can also trigger a full repository scan manually.

Step 4.2: Integrate Amazon Inspector for ECR Scanning

1. Go to the **Amazon Inspector console**.
2. In **Settings -> Scan settings**, ensure that **ECR container image scanning** is enabled.
3. That's it. Inspector will now automatically scan any image pushed to ECR.

Step 4.3: Modify the `buildspec.yml` for SAST & SCA

Update the `buildspec.yml` file from the previous CI/CD project to include new security scanning commands. This example is for a Python application.

```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.9
    commands:
      - pip install bandit # SAST linter for Python
      - pip install safety # SCA for Python dependencies

  pre_build:
    commands:
      - echo Running SAST scans...
      - bandit -r . -f json -o bandit-report.json || true # Run Bandit,
don't fail the build on findings

      - echo Running SCA scans...
      - safety check -r requirements.txt --json > safety-report.json || true
# Run Safety

  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      # ... (Docker build commands as in previous project) ...

  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      # ... (Docker push commands as in previous project) ...

      - echo Importing findings to Security Hub...
      - |-
        python -c '
        import boto3, json, os
        securityhub = boto3.client("securityhub")
        account_id = os.environ["AWS_ACCOUNT_ID"]
        region = os.environ["AWS_REGION"]
        repo_name = os.environ["CODEBUILD_SOURCE_REPO_URL"].split("/")[-1]

        # Parse and import Bandit findings
        with open("bandit-report.json") as f:
            bandit_findings = json.load(f)["results"]
```

```
        formatted_findings = []
        for item in bandit_findings:
            finding = {
                "SchemaVersion": "2018-10-08",
                "Id": f"{repo_name}/{item['filename']}/{item['test_id']}",
                "ProductArn": f"arn:aws:securityhub:{region}:
{account_id}:product/{account_id}/default",
                "GeneratorId": "BanditSAST",
                "AwsAccountId": account_id,
                "Types": ["Software and Configuration
Checks/Vulnerabilities/SAST"],
                "CreatedAt": item['test_run_time'],
                "UpdatedAt": item['test_run_time'],
                "Severity": {"Label": item['issue_severity']},
                "Title": f"Bandit Issue: {item['issue_text']}",
                "Description": f"{item['issue_text']} found in
{item['filename']} at line {item['line_number']}",
                "Resources": [{
                    "Type": "AwsCodeCommitRepository",
                    "Id": f"arn:aws:codecommit:{region}:{account_id}:
{repo_name}",
                    "Details": {"AwsCodeCommitRepository": {"Name":
repo_name}}
                }]
            }
            formatted_findings.append(finding)

        if formatted_findings:
            securityhub.batch_import_findings(Findings=formatted_findings)
            print(f"Imported {len(formatted_findings)} findings from Bandit
to Security Hub.")
        '
artifacts:
  files:
    - imagedefinitions.json
    - bandit-report.json
    - safety-report.json
```

Key Changes:

- **`install` phase:** We install `bandit` (a SAST tool for Python) and `safety` (an SCA tool).
- **`pre_build` phase:** We run the tools and output their findings to JSON files. We use `|| true` to prevent the build from failing if vulnerabilities are found. In a

stricter pipeline, you would remove this and fail the build.

- **`post_build` phase:** This is the most important part. We add a Python script that:
    1. Parses the JSON output from our security tools (in this case, Bandit).
    2. Transforms the findings into the **AWS Security Finding Format (ASFF)**.
    3. Uses the `batch_import_findings` Boto3 API call to send these findings to AWS Security Hub.

Step 4.4: Grant Permissions

- Go to the IAM role used by your CodeBuild project.
- Add the permission `securityhub:BatchImportFindings`.

Step 4.5: Set Up DAST (Conceptual)

1. **Add a new stage** to your CodePipeline after the staging deployment, named `Dynamic-Analysis`.
2. **Create a new CodeBuild project** for this stage.
3. In the `buildspec.yml` for this new project, use commands to run OWASP ZAP against your staging application's URL.

```
# buildspec-dast.yml
version: 0.2
phases:
  install:
    commands:
      - wget
https://github.com/zaproxy/zaproxy/releases/download/v2.11.1/ZAP_2.11.1_L
      - tar -xvf ZAP_2.11.1_Linux.tar.gz
  build:
    commands:
      - ./ZAP_2.11.1/zap.sh -cmd -quickurl $STAGING_APP_URL -
quickprogress -report zap-report.html
artifacts:
  files:
    - zap-report.html
```

Note: This is a simplified example. A real implementation would involve more configuration and potentially importing the ZAP findings to Security Hub as well.

---

5. Viewing the Results

1. **Push a code change** to trigger the pipeline.
2. Let the pipeline run through the build stage.
3. Go to the **AWS Security Hub console -> Findings**.
4. You will see the findings from Bandit (prefixed with `BanditSAST`) and the findings from Amazon Inspector (from the ECR scan) all in one place. You can filter by generator, severity, and other fields.

---

6. Cleanup

1. **Delete the CodePipeline** and its associated CodeBuild projects.
2. **Disassociate the repository** from Amazon CodeGuru.
3. **Disable Amazon Inspector** if desired.
4. **Delete the findings** in Security Hub if you do not want to retain them.
5. Clean up all other resources from the underlying CI/CD pipeline (ECS cluster, ECR repo, etc.).

Business Context

The Problem

Development teams face slow deployment cycles, manual testing bottlenecks, and security vulnerabilities introduced during the CI/CD process. Traditional deployment methods lack security scanning, leading to vulnerabilities reaching production. Manual processes create inconsistency and human error.

The Solution

Secure CI/CD pipeline with integrated security scanning, automated testing, and infrastructure as code. Implements shift-left security with SAST, DAST, and dependency scanning. Automates deployments while enforcing security gates and compliance checks at every stage.

Business Value

- **Faster Time to Market:** Automated pipelines reduce deployment time from weeks to hours
- **Improved Security Posture:** Catches vulnerabilities before production deployment
- **Reduced Manual Effort:** Eliminates 80%+ of manual deployment tasks
- **Audit Trail:** Complete deployment history and security scan results for compliance

Risk Mitigation

Prevents deployment of vulnerable code, hardcoded secrets, misconfigured infrastructure, and non-compliant resources to production environments.

GRC Mapping

Compliance Frameworks

- **NIST CSF:** PR.IP-1 (Baseline configuration), PR.DS-6 (Integrity checking), DE.CM-4 (Code analysis)
- **ISO 27001:** A.12.1 (Operational procedures), A.14.2 (Security in development), A.12.4 (Logging)
- **CIS Controls:** Control 16 (Application Software Security), Control 11 (Secure Configuration)
- **OWASP DevSecOps:** Security testing integration, Secret management, Dependency checking

Security Controls Implemented

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Software Composition Analysis (SCA)
- Secrets scanning and prevention
- Infrastructure as Code security validation

Audit Evidence

- Pipeline execution logs with security scan results
- Code commit and approval records
- Security test reports (SAST/DAST/SCA)
- Deployment approval and rollback records

Regulatory Alignment

- **SOX:** Section 404 (Change management controls)
- **PCI DSS:** Requirement 6.3 (Secure development), Requirement 6.5 (Common vulnerabilities)
- **GDPR:** Article 25 (Data protection by design), Article 32 (Security of processing)
- **SOC 2:** CC8.1 (Change management), CC7.2 (System monitoring)