Sprint 5 - Building a Secure DevOps Pipeline for CI/CD

Team Lead: KJ McDaniels
Project Manager: Luz Ritacco
DevOps Engineer: Zedd Chisholm
DevOps Engineer: Opeyemi Olaleye
Security Engineer: Yonisbel Soto

Continuous Monitoring Integration

1. **Monitoring Architecture Diagram**: Provide a detailed diagram illustrating the integration of application performance monitoring (APM) and infrastructure monitoring tools with the CI/CD pipeline, including the data flow and communication channels.

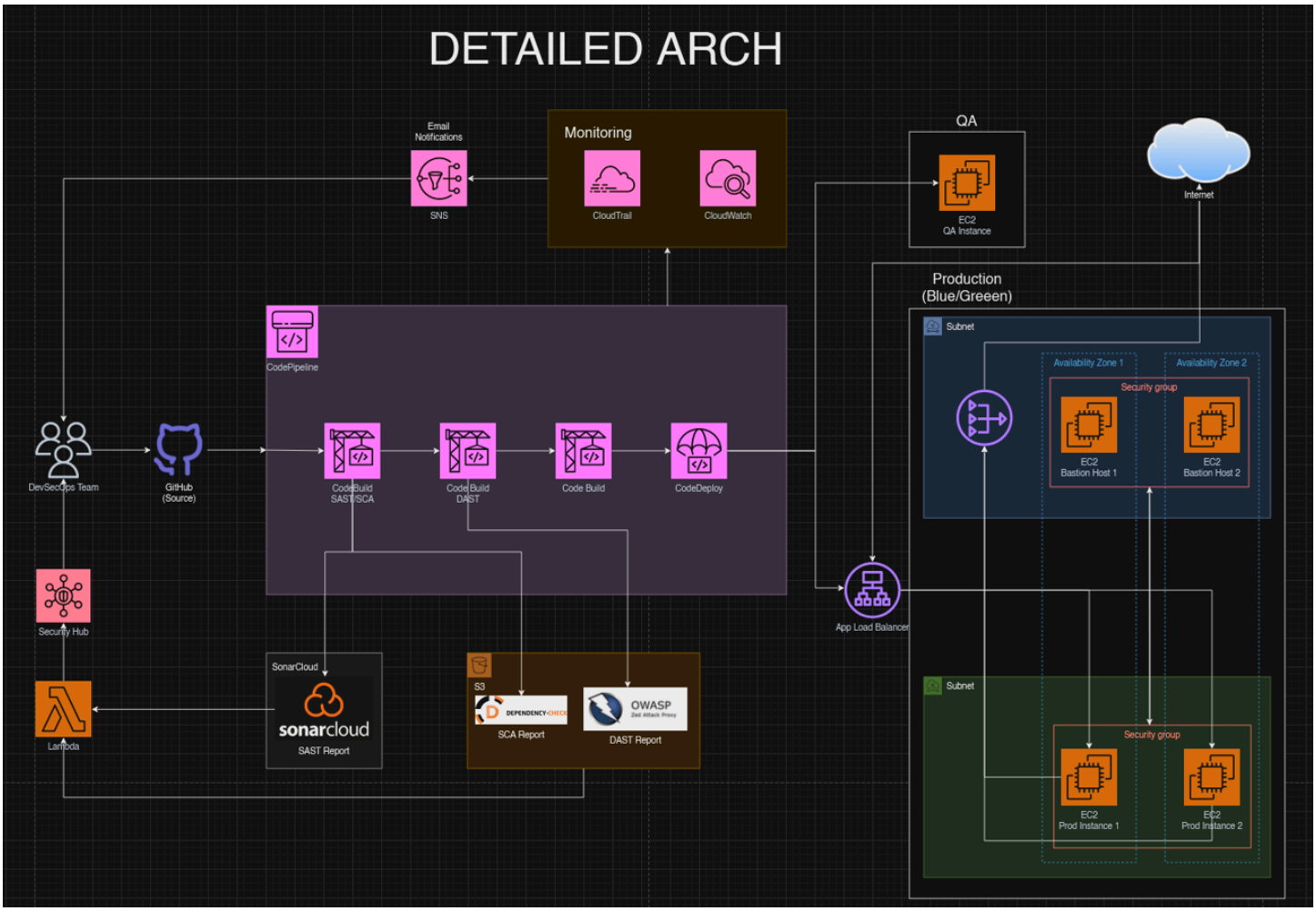


Fig 1: Detailed Architecture (created by Zedd Chisholm)

2. Monitoring Tool Configuration Guide: Document the step-by-step process for configuring and deploying the selected APM and infrastructure monitoring tools, including any necessary agent installations, instrumentation, and integration with the CI/CD pipeline.

Amazon CloudWatch Configuration:

a. Enable CloudWatch agent on EC2 instances:

- SSH into your EC2 instance
- Install the CloudWatch agent:
  - sudo yum install amazon-cloudwatch-agent
- Configure the agent:
  - sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
- Start the agent
  - sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json

b. Set up CloudWatch Logs:

- In the AWS Console, go to CloudWatch
- Click on "Log groups" in the left sidebar
- Click "Create log group"
- Name your log group (e.g., "/aws/ec2/myapp")
- Configure your application to send logs to this group

c. Create CloudWatch Dashboard:

- In CloudWatch, click "Dashboards" then "Create dashboard"
- Name your dashboard
- Add widgets for key metrics (CPU, Memory, Custom metrics)

d. Set up CloudWatch Alarms:

- In CloudWatch, click "Alarms" then "Create alarm"
- Select the metric to alarm on
- Set the threshold and condition for the alarm
- Choose or create an SNS topic for notifications

Amazon Managed Grafana Setup:

a. Create Grafana Workspace:

- Go to Amazon Managed Grafana in the AWS Console
- Click "Create workspace"
- Name your workspace and create it

b. Configure data sources:

- In your Grafana workspace, go to "Configuration" > "Data sources"
- Add CloudWatch as a data source

c. Create dashboards:

- In Grafana, click "Create" > "Dashboard"
- Add panels for visualizing your CloudWatch

Integration with CI/CD Pipeline:

a. Update your application code:

- Add necessary SDK installations and configurations to your application code
- Commit these changes to your repository

b. Modify CI/CD pipeline:

- In your CodeBuild buildspec.yml, add steps to install and configure monitoring agents
- In CodeDeploy, ensure your deployment scripts start the monitoring agents

c. Add post-deployment health checks:

- Create a Lambda function that checks CloudWatch metrics after deployment
- Add this Lambda as a step in your CodePipeline

Set up Alerting and Automation:

a. Create SNS topics for different alert types

b. Configure CloudWatch Alarms to publish to these SNS topics

c. Create Lambda functions for automated responses to specific alerts

d. Set up EventBridge rules to trigger these Lambda functions based on monitoring events

Testing and Validation:

a. Deploy your application through the updated CI/CD pipeline

b. Verify that all monitoring tools are collecting data

c. Test your alerts by deliberately triggering alarm conditions

d. Validate that your dashboards in CloudWatch and Grafana are displaying correct information

3. Monitoring Dashboards and Alerts: Develop and document customized monitoring dashboards and alert configurations tailored to the specific application and infrastructure

components, including performance metrics, health checks, and potential issue indicators.

Identify Key Metrics and Health Indicators:

a. Application Performance Metrics:

- Response time
- Error rate
- Throughput (requests per second)
- Active users
- Database query performance

b. Infrastructure Health Metrics:

- CPU utilization
- Memory usage
- Disk I/O
- Network throughput
- ELB request count and latency

c. Business-specific Metrics:

- Transaction success rate
- User sign-ups
- Revenue-related metrics

Create CloudWatch Dashboards:

a. General System Health Dashboard:

- Open CloudWatch in AWS Console
- Click "Dashboards" then "Create dashboard"
- Name it "System Health Overview"
  - Add widgets for:
    - EC2 CPU Utilization (Line graph)
    - EC2 Memory Usage (Line graph)
    - ELB Request Count (Line graph)
    - ELB Latency (Line graph)
    - RDS Database Connections (Line graph)

b. Application Performance Dashboard:

- Create a new dashboard named "Application Performance"
- Add widgets for:
  - Application Error Rate (Line graph)
  - API Response Times (Line graph)
- Active Users (Number widget)
  - Successful Transactions (Number widget)
  - Top 5 Slowest API Endpoints (Table widget)

c. Business Metrics Dashboard:

- Create a new dashboard named "Business Metrics"
- Add widgets for:
  - Daily New User Sign-ups (Bar graph)
  - Revenue per Hour (Line graph)
  - Active Subscriptions (Number widget)
  - Customer Churn Rate (Gauge widget)

Set Up CloudWatch Alarms:

a. High CPU Utilization Alarm:

- In CloudWatch, click "Alarms" then "Create alarm"
- Select EC2 > Per-Instance Metrics > CPUUtilization
- Set threshold to greater than 80% for 5 minutes
- Create an SNS topic for "High CPU Alerts" and add it as an action

b. High Error Rate Alarm:

- Create a custom metric for application error rate if not exists
- Create an alarm when error rate is greater than 5% for 5 minutes
- Use the same SNS topic or create a new one for "Application Alerts"

c. Low Free Storage Space Alarm:

- Select the appropriate EBS volume metric
- Set alarm when free storage is less than 20% for 30 minutes
- Add to "Infrastructure Alerts" SNS topic

d. Abnormal User Activity Alarm:

- Create a custom metric for user logins per minute
- Set up an anomaly detection alarm with a band of 2 standard deviations
- Alert when outside this band for 15 minutes
- Add to "Security Alerts" SNS topic

Set Up Grafana Dashboards (if using Amazon Managed Grafana):

a. Infrastructure Overview:

- Create a new dashboard
- Add panels for:
  - EC2 instance health status
  - ECS cluster capacity
  - RDS database performance

b. Application Deep Dive:

- Create panels for:

    - API gateway metrics
    - Lambda function performance
    - Custom application metrics

Document Dashboard and Alert Configurations:

a. Create a "Monitoring Runbook" document:

- List all dashboards with their purposes
- Document each widget in the dashboards and what it represents
- List all configured alarms with their thresholds and actions
- Include screenshots of key dashboards for quick reference

b. Alert Response Procedures:

- For each type of alert, document:
    - What the alert means
    - Initial diagnostic steps
    - Escalation procedures
    - Potential remediation actions

Implement Continuous Improvement Process:

a. Schedule regular reviews of dashboards and alerts

b. Gather feedback from team members on dashboard usability

c. Analyze past incidents to identify any missing alerts or metrics

d. Update dashboards and alerts based on application changes or new features

Security Auditing & Vulnerability Scanning

1. **Vulnerability Scanning Tool Integration Guide**:

This guide details the process of integrating vulnerability scanning tools within the CI/CD pipeline, covering Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), and container scanning. It includes configurations, scheduling scans, and reporting mechanisms.

Description: Implement and configure static code analysis (SCA) tools within the CI/CD pipeline to automatically scan and identify vulnerabilities or code quality issues in the early stages of development. This setup ensures that the codebase adheres to predefined quality standards and security benchmarks before proceeding to the build stage.

Static Application Security Testing (SAST): SAST tools like SonarQube can be integrated into the CI pipeline to analyze the source code and detect potential vulnerabilities, coding flaws,

and security weaknesses early in the development process.

Dynamic Application Security Testing (DAST): DAST tools like OWASP ZAP can be used to perform web application security testing by simulating real-world attacks and identifying vulnerabilities such as cross-site scripting (XSS), SQL injection, and other common web application threats. These tests can be triggered during the deployment phase or as part of a separate testing stage in the pipeline.

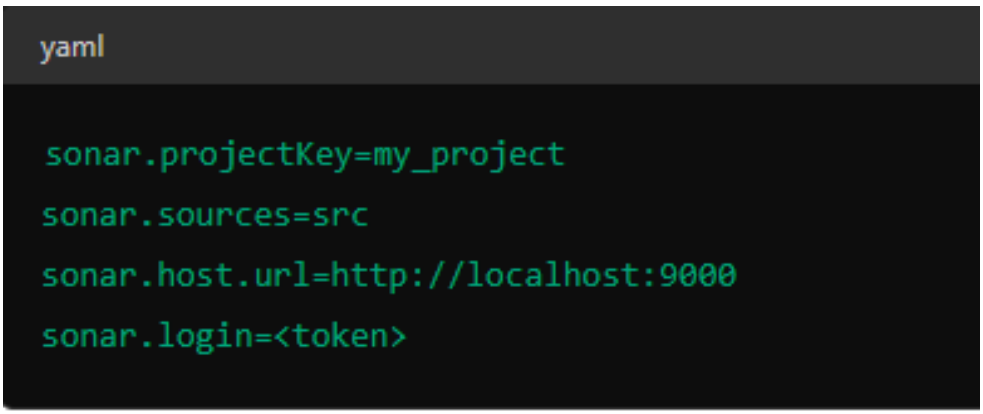
Software Composition Analysis (SCA): SCA tools like AWS CodeArtifact, Snyk, can be used to scan and analyze third-party libraries and dependencies for known vulnerabilities and licensing issues. This helps identify and mitigate risks associated with using open-source components in your application.

Tools

- **SAST Tools**: SonarQube, Fortify, Checkmarx
- **DAST Tools**: OWASP ZAP, Burp Suite, Arachni
- **Container Scanning Tools**: Trivy, Clair, Aqua Security

Integration Steps

1. **Tool Configuration**
    - **SonarQube (SAST)**
        1. Install SonarQube server and SonarScanner.
        2. Configure SonarQube



3. Add SonarScanner to your CI/CD pipeline (e.g., Jenkins, GitLab CI):

```yaml
stage('SAST') {
    steps {
        script {
            sh 'sonar-scanner'
        }
    }
}
```

- **OWASP ZAP (DAST)**

1. Install OWASP ZAP and configure API.
2. Add OWASP ZAP to CI/CD pipeline

```yaml
stage('DAST') {
    steps {
        script {
            sh 'zap-baseline.py -t http://myapp:8080'
        }
    }
}
```

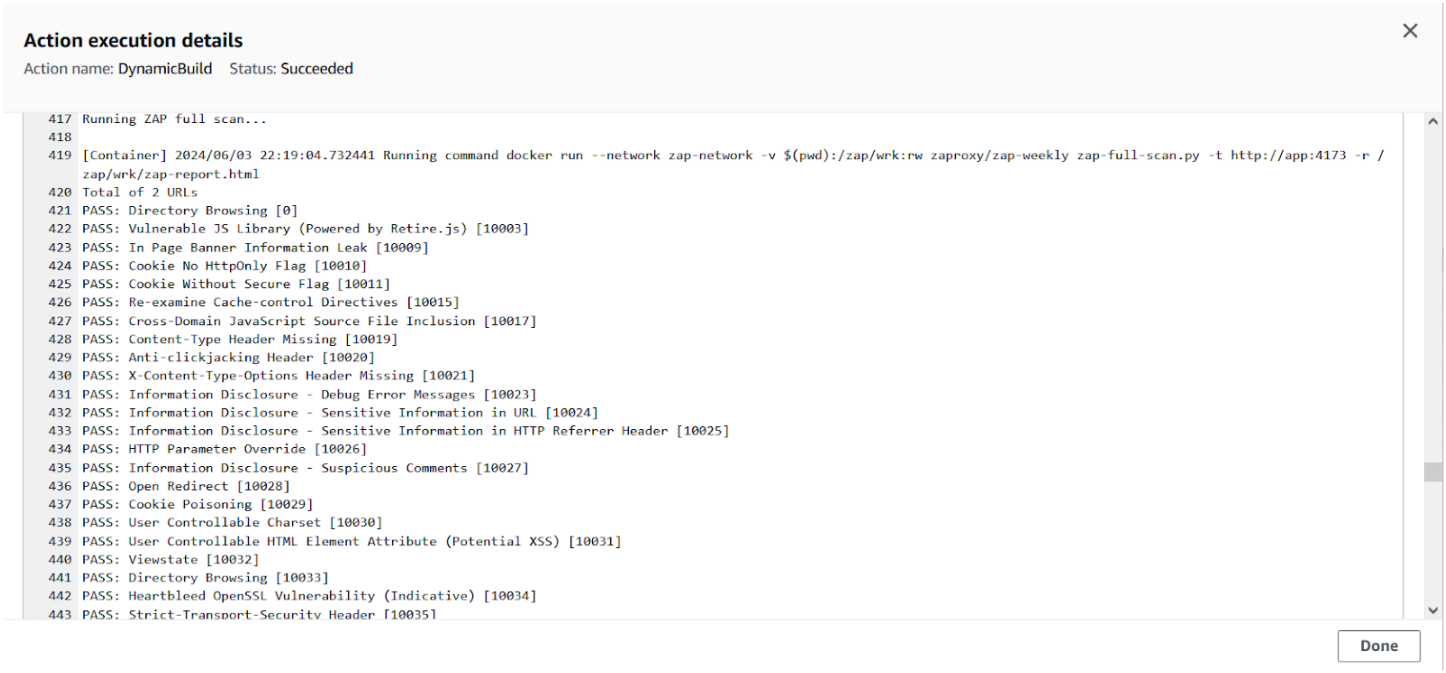
2. Scan Scheduling

- **On Code Commit/Push**: Integrate scans to trigger on each code commit or push to the repository.
- **Nightly Builds**: Schedule scans to run as part of nightly build jobs to ensure comprehensive coverage.
- **Pre-Deployment**: Run scans as a gating mechanism before deploying to production.

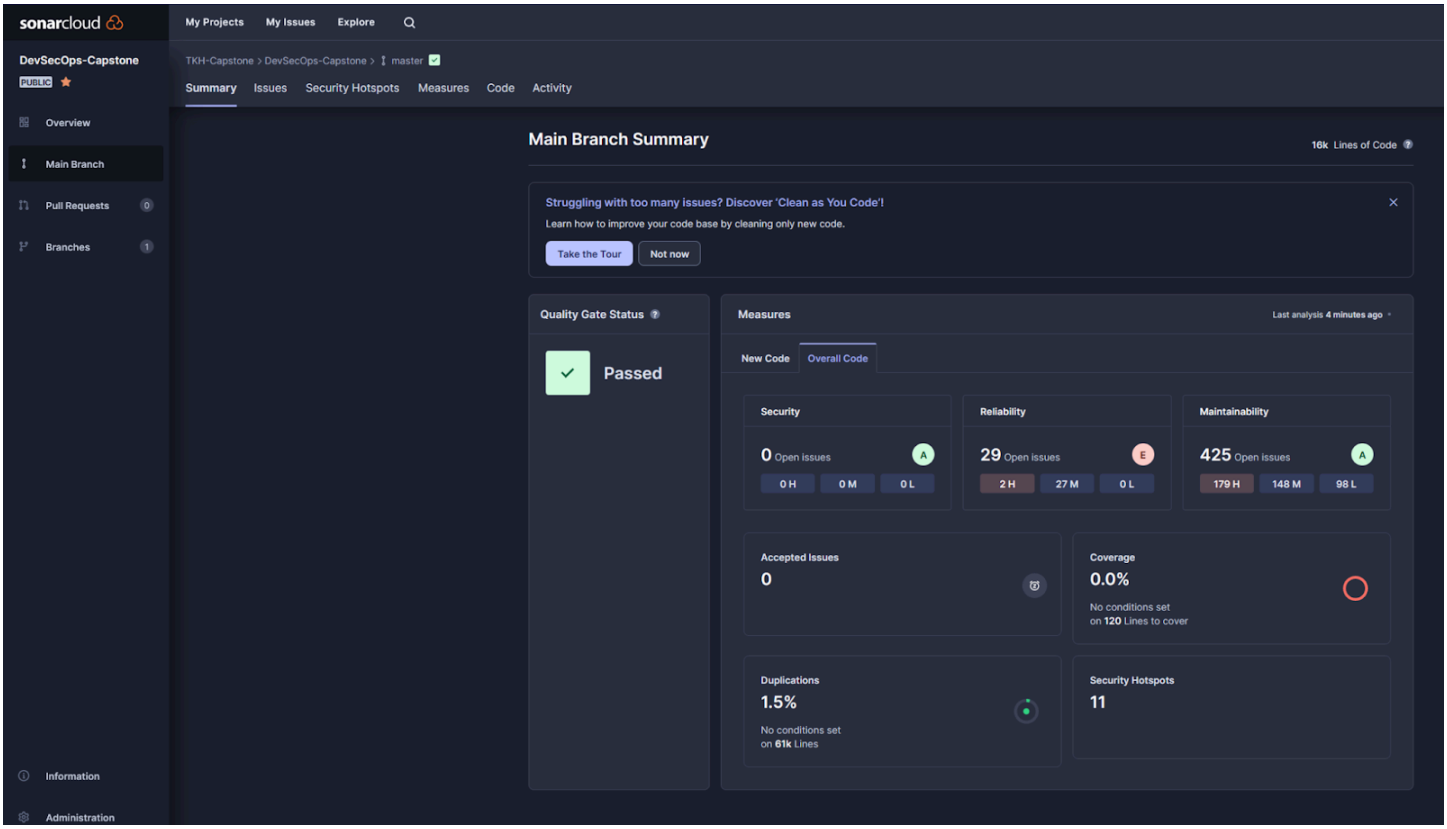
3. Reporting Mechanisms

- **SonarQube**
  - View results in SonarQube dashboard.

- Set up email notifications for new vulnerabilities.



- Successfully integrated Sonarqube in the pipeline.



- Successfully integrated dependency check into the QA branch.

DEPENDENCY-CHECK

How to read the report | Suppressing false positives | **Getting Help:** github issues

Sponsor

Project: MyProject

Scan Information (show all):

- *dependency-check version*: 9.2.0
- *Report Generated On*: Mon, 27 May 2024 05:15:02 GMT
- *Dependencies Scanned*: 4894 (3306 unique)
- *Vulnerable Dependencies*: 15
- *Vulnerabilities Found*: 34
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence
@babel/traverse:7.21.5		pkg:npm/%40babel%2Ftraverse@7.21.5	CRITICAL	1		8
axios:0.26.1		pkg:npm/axios@0.26.1	MEDIUM	2		3
dependency-check-9.2.0-release.zip: bcpg-jdk18on-1.71.jar	cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.71:*:*:*:*:*:*:* cpe:2.3:a:openpgp:openpgp:1.71:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcpg-jdk18on@1.71	MEDIUM	1	Highest	60
dependency-check-9.2.0-release.zip: bcprov-jdk18on-1.71.jar	cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.71:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:the_bouncy_castle_crypto_package_for_java:1.71:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcprov-jdk18on@1.71	HIGH	5	Highest	60
dependency-check-9.2.0-release.zip: h2-2.1.214.jar	cpe:2.3:a:h2database:h2:2.1.214:*:*:*:*:*:*:*	pkg:maven/com.h2database/h2@2.1.214	HIGH	2	Highest	45
dependency-check-9.2.0-release.zip: logback-core-1.2.11.jar	cpe:2.3:a:qos:logback:1.2.11:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.2.11	HIGH	1	Highest	32
express:4.18.2		pkg:npm/express@4.18.2	MEDIUM	2		3
follow-redirects:1.15.2		pkg:npm/follow-redirects@1.15.2	MEDIUM	4		3
postcss:8.4.23	cpe:2.3:a:postcss:postcss:8.4.23:*:*:*:*:*:*:*	pkg:npm/postcss@8.4.23	MEDIUM	2	Highest	8
semver:6.3.0		pkg:npm/semver@6.3.0	HIGH	2		5
sonar-scanner-cli-5.0.1.3006.jar (shaded: com.squareup.okhttp3:okhttp-urlconnection:3.14.2)	cpe:2.3:a:squareup:okhttp:3.14.2:*:*:*:*:*:*:* cpe:2.3:a:squareup:okhttp3:3.14.2:*:*:*:*:*:*:*	pkg:maven/com.squareup.okhttp3/okhttp-urlconnection@3.14.2	MEDIUM	1	Highest	9
sonar-scanner-cli-5.0.1.3006.jar (shaded: com.squareup.okhttp3:okhttp:3.14.2)	cpe:2.3:a:squareup:okhttp:3.14.2:*:*:*:*:*:*:* cpe:2.3:a:squareup:okhttp3:3.14.2:*:*:*:*:*:*:*	pkg:maven/com.squareup.okhttp3/okhttp@3.14.2	HIGH	2	Highest	9
sonar-scanner-cli-5.0.1.3006.jar (shaded: com.squareup.okio:okio:1.17.2)	cpe:2.3:a:squareup:okio:1.17.2:*:*:*:*:*:*:*	pkg:maven/com.squareup.okio/okio@1.17.2	HIGH	1	Highest	9
vite:4.3.3	cpe:2.3:a:vitejs:vite:4.3.3:*:*:*:*:*:*:*	pkg:npm/vite@4.3.3	HIGH	6	Highest	8

2.Vulnerability Management Workflow

Workflow Overview

1. **Identification**
   - Automated tools identify vulnerabilities during CI/CD pipeline execution.
   - Security team reviews automated scan results.
2. **Triage**
   - Assess severity and impact of identified vulnerabilities.
   - Classify vulnerabilities based on risk (Critical, High, Medium, Low).
3. **Prioritization**
   - Prioritize remediation efforts based on vulnerability severity and impact on the business.

- Create tickets in the issue tracking system (e.g., Jira) for each identified vulnerability.

4. **Remediation**
   - Development team addresses identified vulnerabilities by fixing code, updating dependencies, or reconfiguring environments.
   - Operations team applies necessary patches and configuration changes to infrastructure.
5. **Verification**
   - Re-run scans to verify that vulnerabilities have been remediated.
   - Conduct manual security testing if needed.
6. **Reporting**
   - Generate and distribute regular vulnerability status reports to stakeholders.
   - Document remediation steps and maintain a vulnerability management log.

Site: http://app:8000

Generated on Tue, 11 Jun 2024 23:11:45

ZAP Version: D-2024-06-10

ZAP is supported by the Crash Override Open Source Fellowship

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	0
Low	0
Informational	0
False Positives:	0

Roles and Responsibilities

- **Development Team**
  - Fix code-level vulnerabilities.
  - Implement secure coding practices.
- **Security Team**
  - Review scan results and triage vulnerabilities.
  - Provide guidance on remediation.
- **Operations Team**
  - Apply patches and update infrastructure configurations.
  - Monitor for compliance with security policies.

3.Security Audit Checklist

Secure Coding Practices

- Ensure input validation and sanitization.
- Implement proper error handling and logging.
- Use parameterized queries to prevent SQL injection.
- Secure API endpoints with proper authentication and authorization mechanisms.

Infrastructure Hardening

- Ensure all servers and services are up-to-date with security patches.
- Disable unnecessary services and ports.
- Use firewalls and intrusion detection/prevention systems.
- Implement network segmentation.

Compliance with Security Policies and Standards

- Adhere to industry standards (e.g., OWASP Top 10, CIS Benchmarks).
- Ensure data encryption at rest and in transit.
- Conduct regular security training for all team members.
- Perform regular security audits and penetration testing.

CI/CD Pipeline Security

- Ensure the CI/CD pipeline is secure and only accessible to authorized personnel.

Security Auditing & Vulnerability Scanning

In a CI/CD DevSecOps pipeline, continuous integration and continuous delivery/deployment are integrated with security at every stage. This ensures that security and performance feedback are reviewed and incorporated continuously. Below is a structured process for achieving this, including risk assessment, prioritization, and action planning.

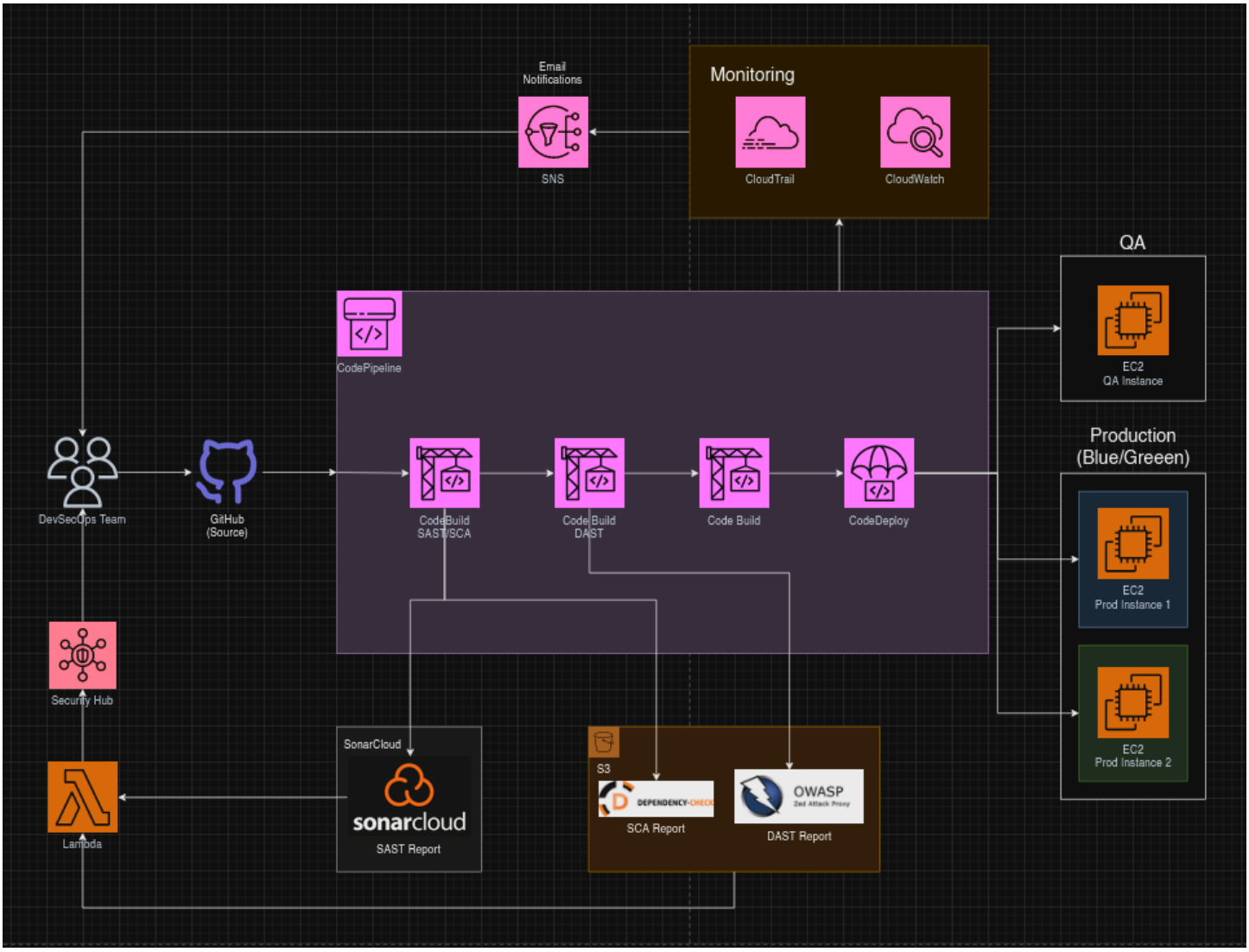


Fig 8: Feedback Loop Architecture (created by Zedd Chisholm)

1. Feedback Collection

Security Feedback

- **Static Application Security Testing (SAST):** Analyze source code for security vulnerabilities using SonarQube.
- **Dynamic Application Security Testing (DAST):** Test the running application for security issues using OWASZAP.
- **Software Composition Analysis (SCA):** Identify vulnerabilities in third-party libraries and dependencies.
- **Secret Detection Tools:** Tools like GitGuardian to identify hard-coded secrets in the codebase.

Performance Feedback

- **Performance Monitoring:** Continuous monitoring of the application's performance metrics.

2. Feedback Review and Risk Assessment

Feedback Review

- **Automated Reports:** Generate automated reports from security and performance tools.
- **Continuous Monitoring:** Utilize dashboards for real-time monitoring of security and performance metrics.

Risk Assessment

- **Severity Analysis:** Classify issues based on severity (Critical, High, Medium, Low).
- **Impact Analysis:** Assess the potential impact on the system, data, and users.
- **Likelihood Analysis:** Determine the likelihood of the issue being exploited or occurring.

3. Prioritization

Criteria for Prioritization

- **Business Impact:** Prioritize issues that impact critical business functions.
- **User Impact:** Focus on issues affecting the user experience.
- **Compliance Requirements:** Address issues related to regulatory compliance first.
- **Resource Availability:** Consider the availability of resources to address the issues.
- **Weighted Scoring:** Assign weights to different criteria (e.g., severity, impact, compliance) and score issues accordingly.

4. Action Planning

Planning

- **Define Objectives:** Set clear objectives for addressing the identified issues.
- **Assign Responsibilities:** Assign tasks to appropriate team members (developers, security team, ops team).
- **Set Deadlines:** Establish realistic timelines for remediation based on prioritization.

Implementation

- **Develop Fixes:** Implement code changes, patches, or configuration updates to address issues.
- **Test Fixes:** Rigorously test the fixes in a staging environment before deployment.
- **Deploy Fixes:** Use the CI/CD pipeline to deploy the fixes to production.

Verification

- **Post-Deployment Testing:** Conduct post-deployment testing to ensure the issues have been resolved.
- **Continuous Monitoring:** Monitor the application for any regression or new issues.

5. Continuous Improvement

Retrospective Analysis

- **Post-Mortem Analysis:** Conduct post-mortem analysis for critical issues.
- **Root Cause Analysis (RCA):** Identify the root causes of issues to prevent recurrence.

Process Improvement

- **Update Policies:** Update security and performance policies based on lessons learned.
- **Enhance Tools:** Integrate new tools or update existing tools to improve detection and remediation.
- **Training and Awareness:** Conduct training sessions for the development and operations teams on best practices and new policies.

Example Workflow

1. **Trigger:** New code push or pull request triggers CI/CD pipeline.
2. **Automated Testing:** Run SAST, DAST, and performance tests.
3. **Feedback Collection:** Collect results and generate reports.
4. **Review:** Automated tools and manual review by security and performance teams.
5. **Risk Assessment:** Evaluate severity, impact, and likelihood.
6. **Prioritization:** Use risk matrix and weighted scoring.
7. **Action Plan:** Develop, test, and deploy fixes.
8. **Verification:** Post-deployment testing and monitoring.
9. **Continuous Improvement:** Conduct retrospectives and update processes.

Knowledge Transfer & Handover Package

1. **CI/CD Pipeline Documentation**: Create comprehensive documentation covering the entire CI/CD pipeline, including the tools, configurations, and processes involved in building, testing, and deploying applications securely
    - https://www.canva.com/design/DAGI5Kya1L4/XvcRGfq3Olv2rPogEo76Pg/edit
2. **Training Materials**: Develop training materials, such as presentations, videos, or interactive tutorials, to facilitate knowledge transfer to developers and operations teams, covering topics like secure coding practices, vulnerability management, and continuous monitoring.

    Group Presentation

    - https://www.canva.com/design/DAGCIXpqAFQ/0PcW4HScr93AALiD4R4XGw/edit

3**. Source Code Documentation**: Document the source code for any custom scripts, tools, or automation developed during the project, including inline comments, architecture diagrams, and API documentation.

- Backend

    - https://github.com/KJMcDaniels/DevOps-Pipeline-for-CI-CD/blob/715a3f4d539f1465bd43bfc270f0db245d27db89/backend/index.js#L19
- Buildspec
    - https://github.com/KJMcDaniels/DevOps-Pipeline-for-CI-CD/blob/715a3f4d539f1465bd43bfc270f0db245d27db89/buildspec.yml#L6
- Buildspec-DAST
    - https://github.com/KJMcDaniels/DevOps-Pipeline-for-CI-CD/blob/715a3f4d539f1465bd43bfc270f0db245d27db89/buildspec-dast.yml#L5
- Buildspec-QA
    - https://github.com/KJMcDaniels/DevOps-Pipeline-for-CI-CD/blob/715a3f4d539f1465bd43bfc270f0db245d27db89/buildspec-qa.yml#L6

Handover Checklist: Prepare a handover checklist to ensure a smooth transition of the secure CI/CD pipeline to the development and operations teams, including outstanding tasks, **known issues**, and future enhancement plans.

HANDOVER CHECKLIST

1. Documentation
   - [ ] Architecture diagram
   - [ ] Pipeline workflow documentation
   - [ ] Security measures and controls
   - [ ] AWS services configuration guide
   - [ ] Deployment processes
   - [ ] Rollback procedures

2. Access and Permissions
   - [ ] AWS IAM user accounts for team members
   - [ ] Role-based access control (RBAC) setup
   - [ ] Multi-factor authentication (MFA) enabled
   - [ ] SSH keys for relevant services
   - [ ] API keys and secrets (ensure secure storage)

3. Source Code Management
   - [ ] Repository access granted to team members
   - [ ] Branch protection rules configured
   - [ ] Code review process documented
   - [ ] Merge/pull request templates created

4. CI/CD Pipeline
   - [ ] Pipeline configuration files
   - [ ] Build scripts and Dockerfiles
   - [ ] Test suites and automation scripts
   - [ ] Deployment scripts and templates

- [ ] Environment-specific configurations

5. Monitoring and Logging
   - [ ] CloudWatch dashboards and alarms
   - [ ] Log aggregation and analysis setup
   - [ ] Metrics collection and visualization
   - [ ] Alerting mechanisms and escalation procedures

6. Security
   - [ ] Vulnerability scanning tools integration
   - [ ] Secret management solution (e.g., AWS Secrets Manager)
   - [ ] Infrastructure-as-Code (IaC) security checks
   - [ ] Compliance and audit logging

7. Knowledge Transfer
   - [ ] Training sessions scheduled
   - [ ] Runbooks for common operations
   - [ ] Troubleshooting guides
   - [ ] Contact list for support and escalation

8. Outstanding Tasks
   - [ ] List of pending bug fixes
   - [ ] Incomplete features or enhancements
   - [ ] Technical debt items

9. Known Issues
   - [ ] Documented workarounds for known bugs
   - [ ] Performance bottlenecks
   - [ ] Compatibility issues

10. Future Enhancement Plans
    - [ ] Roadmap for upcoming features
    - [ ] Scalability improvements
    - [ ] Integration with additional services
    - [ ] Automation opportunities

11. Third-party Integrations
    - [ ] List of integrated services
    - [ ] API documentation and credentials
    - [ ] Service level agreements (SLAs)

12. Cost Management
    - [ ] AWS cost allocation tags
    - [ ] Budget alerts and reports

- [ ] Resource optimization recommendations

13. Compliance and Governance
    - [ ] Relevant compliance standards (e.g., SOC 2, HIPAA)
    - [ ] Audit trails and reports
    - [ ] Policy enforcement mechanisms

14. Performance Benchmarks
    - [ ] Current performance metrics
    - [ ] Load testing results
    - [ ] Capacity planning guidelines

Operational Runbooks: Develop operational runbooks detailing the procedures for maintaining, **troubleshooting**, and scaling the secure CI/CD pipeline, incident response procedures, and capacity planning guidelines.

OPERATIONAL RUNBOOK

1. Maintenance Procedures:
   - Daily: Monitor pipeline health, review logs
   - Weekly: Update dependencies, run security scans
   - Monthly: Perform full system backup, review access controls
2. Troubleshooting:
   - Pipeline Failures:
     i. Check AWS CloudWatch logs
     ii. Verify IAM permissions
     iii. Ensure resource availability
- Deployment Issues:
     i. Validate AWS CodeDeploy configurations
     ii. Check EC2 instance health
     iii. Review application logs
3. Scaling:
   - Monitor CPU/memory usage with CloudWatch
   - Set up Auto Scaling groups for EC2 instances
   - Use AWS Lambda for serverless components
   - Implement Amazon EKS for container orchestration
4. Incident Response:
   - Classify severity (Low, Medium, High, Critical)
   - Isolate affected systems
   - Investigate root cause
   - Implement fix and verify
   - Post-incident review and update procedures
5. Capacity Planning:
   - Review CloudWatch metrics monthly
   - Forecast growth based on historical data
   - Plan infrastructure upgrades quarterly

- Consider reserved instances for cost optimization