SonarQube for Jenkins and BitBucket Integration

📌 Architecture Components Overview

Component	Purpose
EC2 (RHEL)	Host for Jenkins & SonarQube
SonarQube	Code quality & security analysis
BitBucket	Source code repository
Jenkins	Executes pipeline & triggers Sonar scans
Database	SonarQube uses a DBMS to store results

Overview

Using SonarQube requires a web server running the SonarQube web-based software, which is used to store results from a scan of source code for an application. Each application requires a unique identifier, and this is used to store and display results, via the SonarQube web-based interface. The actual scanning is accomplished using the SonarQube scanner, which can be invoked using common build tools within a CI/CD pipeline. Results of scans are available via the web-based interface.

FPAC Implementation Summary

SonarQube Web Based Interface

SonarQube is installed on the same EC2 server that is running Jenkins. The SonarQube web interface will be available as a URL, reachable from the USDA VPN. This will allow access to scan reports, generated by SonarQube. This is also required to configure the application id's within SonarQube which are used to record the scans of the data pipelines.

SonarQube Scanning

The SonarQube scanner will be available on the Jenkins server, configured in a manner that will allow the sonar-scanner to be utilized in a Jenkins CI/CD pipeline.

Jenkins Web Hook

There will be a web hook URL, that will be hosted on the Jenkins server, which will allow the integration of BitBucket git events, with Jenkins CI/CD pipelines.

BitBucket Integration

BitBucket git events, such as committing code changes or approving a pull request, will invoke the Jenkins web hook. The call to the web hook will contain the id for the code commit, which is used to pull the code to the Jenkins server from BitBucket.

Jenkins CI/CD Pipeline

The pipeline is called via a web hook; this will provide the commit identifier which is used to pull the correct version of the source code from BitBucket. The Jenkins CI/CD pipeline will run the sonar-scanner tool, with parameters related to the data pipeline code that is being scanned. The example below is sonar-scanner command, if the source code from BitBucket is available from the current directory used in the Jenkins CI/CD pipeline.

Example configuration for sonar-scanner

```
        sonar-scanner \
            -Dsonar.projectKey=carsdatapipeline \
            -Dsonar.sources=. \
            -Dsonar.host.url=http://<USDA FPAC SonarQube Address>:9000 \
            -Dsonar.token=cars-data-pipeline-sonar-qube-token
```

Jenkins access to BitBucket source code

The connection from Jenkins to BitBucket for pulling down the source code is critical. Documentation on this topic varies across tooling and plugins. What is essential is that the commit id provided by the web hook will result in a transfer of the source code to Jenkins. This is essential to the success of running the SonarQube scan. It is assumed that networking changes are required to enable this functionality.

BitBucket access to Jenkins web hook

The connection between the Jenkins server and the BitBucket server, must allow traffic for the web hook. It is assumed that networking changes are required to enable this functionality.

Jenkins CI/CD pipeline access to SonarQube URL

The SonarQube URL will allow the scanning tool to post the results of the scan, Jenkins CI/CD pipeline must be able to access this URL.

SonarQube URL

The ability for users to access the SonarQube URL is critical for both configuring SonarQube and providing access to reports that provide the results of the scans.

SonarQube users

The best approach for user integration with SonarQube web UI is a SAML based solution that integrates with the USDA PIV/CAC login. The level of effort for this integration is not known. The bare minimum is that some number of local users must be created on the SonarQube server, to allow access to reports and configure the application scans.

FPAC Implementation Steps

Install SonarQube Software

Ensure that the SonarQube software is installed and available on the Jenkins server. The SonarQube community edition is a free download, available from this url: https://www.sonarsource.com/products/sonarqube/downloads/success-download-community-edition/

The scanner software for RHEL linux: wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-6.1.0.4477-linux.zip

Create Jenkins Web Hook

Create a web hook on Jenkins, that can be accessed by BitBucket.

Configure BitBucket

Add the Jenkins web hook to BitBucket and associate the web hook with an appropriate git event.

Configure SonarQube for each Data Pipeline

Using the SonarQube web UI, create the project and project token for each data pipeline in BitBucket. The data in the web hook, will contain the BitBucket project name, this should be the same as the SonarQube project name. The configuration data, as displayed in the example above, should be kept in a file in the root folder, within the project repository called: sonar-qube.sh. This file should have execute permissions and will be used to run the sonar-scanner for that specific repository, with the proper configuration data. Using this approach will allow the code to be scanned and results posted for the correct repository, using a single Jenkins CI/CD deployment pipeline for all the data pipelines.

Enable all required network traffic

Enable the ability of the BitBucket server to call the Jenkins web hook. Enable the Jenkins server to access the source code from the BitBucket server. Enable the Jenkins CI/CD pipeline to access SonarQube URL.

Enable users to access SonarQube UI

Users must have access to the SonarQube UI to provision data pipelines to scan and well as to access results of the scans.

FPAC Implementation Challenges and Risks

RISK: CI/CD Pipeline does not deploy software in any environment

The lack of an automated CI/CD process is a critical risk for the SonarQube implementation. An automated CI/CD implementation has been defined, but the well-defined and critical implementation steps have **NOT** been implemented by OCIO and OCIO-DISC teams. The lack of this required implementation leads to a significant gap in assuring that source code scanning is an effective risk reduction strategy. Since there is no automated deployment of software, there should be **no** assurance that the scanned source code is the source code being utilized in any AWS environment. The source code is currently manually deployed in every AWS environment; this represents a significant risk.

RISK: Release process and quality gate risks

The lack of an automated CI/CD deployment process is the root cause of not implementing strict quality gates for source code deployment in all AWS environments. The lack of automation means that there is no assurance that source code tested or scanned in any environment, is the software deployed in production. This is a significant risk. There is a well-defined release process, which can't be implemented without an automated CI/CD process. This type of process reflects both the state of the art and widely accepted best practice for all source code deployments. The lack of implementing standard quality gates, via an automated process, represents a significant risk.

RISK: Developers have AWS read/write console access in production

The lack of an automated CI/CD process leads to a situation that requires developers to have the ability to make source code changes directly to production AWS environment. Generally, this is viewed as an unacceptable risk across most projects, since it violates best practices that are normally implemented via tightly controlled and automated CI/CD source code deployments. This risk appears to far exceed any risk that is mitigated by performing a security scan on source code.