**Source Code Analysis /Static Code Analysis:**

It is a Software verification process in which the source code is analysed for security. It Diagnose the various types of error such as division by zero, run-time errors. It is also a part of Build process which can find a bug early in the development cycle.

**Advantages:**

* It scans all codes in the project and check for the vulnerabilities which is not used.
* Finding errors in the initial stage of development.

**Source Code Analysis Tools:**

* + SonarQube
  + Eclipse
  + FindBugs

**Steps to work on SonarQube:**

**Step 1: Requirement for configuring for SCA.**

In order to configure the SonarQube initially we require a JDK version 1.8.

**Step2**: **Installation process of SonarQube.**

Download the SonarQube from the Enterprise Edition and Unzip the Downloaded folder.

From the Command prompt go to the particular path of bin folder which is inside the SonarQube folder and Execute “StartSonar.bat “command

**Step** **3: Execution on server.**

Once the server gets started we can run SonarQube Server by the URL [http://localhost:9000](http://localhost:9000/)

It takes with default credentials as admin/admin.

**Step 4: Installation process of SonarQube-Scanner.**

To analyse a Project we require to download sonarqube-scanner-cli zip file, and unzip the Downloaded file in the specific path.

Go to the bin folder of sonarqube-scanner folder and execute “sonar-scanner.bat” command to run Batch file.

**Step 5: Work flow of SonarQube.**

* Developers code in their IDEs and use [SonarLint](https://sonarlint.org/) to run local analysis.
* Developers push their code into their favourite SCM: git, SVN.
* The Continuous Integration Server triggers an automatic build, and the execution of the SonarScanner required to run the SonarQube analysis.
* The analysis report is sent to the SonarQube Server for processing.
* SonarQube Server processes and stores the analysis report results in the SonarQube Database, and displays the results in the UI.
* Developers review, comment, challenge their Issues to manage and reduce their rework through the SonarQube UI.

**Step 6: Efficiency of SonarQube.**

SonarQube enables you to centralize and scale a single vision of code quality.

It offers a central place to view and define the rules used during analysis of projects. These rulesets are organized in quality profiles

Ensure fast, efficient report generation and processing across multiple fields of analysis, no matter how complex the project might be.

**Step 7: Analysing the sonarqube-scanner with Maven**

Log on to Jenkins as a Administrator and click on **Manage Jenkins** -> **Global Configuration.**

Scroll down the page to SonarQube servers and enable the “**Enable injection of SonarQube server configuration as build environment variables “.**

**Step 8: Analysing in a Jenkins Pipeline.**

It provides a **‘withSonarQubeEnv’** block that allow the sonarqube server which we want to interact with.

The “waitForQualityGate” step pause the pipeline until SonarQube analysis is completed and returns quality gate status

**Wait for Quality Gate with maven pipeline block**

pipeline {

    agent any

    stages {

        stage('SCM') {

            steps {

                git url: '[respected git url'](https://github.com/foo/bar.git')

            }

        }

        stage('build && SonarQube analysis') {

            steps {

                withSonarQubeEnv('My SonarQube Server') {

                    withMaven(maven:'Maven 3.5') {

                        sh 'mvn clean package sonar:sonar'

                    }

                }

            }

        }

        stage("Quality Gate") {

            steps {

                timeout(time: 1, unit: 'HOURS') {

                    waitForQualityGate abortPipeline: true

                }

            }

        }

    }

}