Analyzing with SonarQube Scanner for Jenkins

Created by Unknown User (hartmann.eric@gmail.com), last modified by Alexandre Gigleux on Jun 28, 2017

By SonarSource – GNU LGPL 3 – Issue Tracker – Sources

SonarQube Scanner for Jenkins 2.6.1

Table of Contents

Features

This plugin lets you centralize the configuration of SonarQube server connection details in Jenkins global configuration.

Then you can trigger SonarQube analysis from Jenkins using standard Jenkins Build Steps to trigger analysis with:

- SonarQube Scanner
- SonarQube Scanner for Maven
- SonarQube Scanner for MSBuild

Once the job is complete, the plugin will detect that a SonarQube analysis was made during the build and display a badge and a widget on the job page with a link to the SonarQube dashboard as well as quality gate status.

SINCE 2.5 : you can also use Jenkins Pipeline DSL (some features require SonarQube >= 6.2).

Compatibility

SonarQube Scanner for Jenkins	2.1	2.2.1	2.3	2.4	2.5 - 2.6
Jenkins	1.491+	1.580.1+	1.587+	1.587+	1.651+

SINCE 2.5 Analysis must run with a JRE8

Installation

1. Install the SonarQube Scanner for Jenkins via the Jenkins Update Center.
2. Configure your SonarQube server(s)
   a. Log into Jenkins as an administrator and go to **Manage Jenkins > Configure System**:
   b. Scroll down to the **SonarQube configuration** section, click on **Add SonarQube**, and add the values you're prompted for.

Use

Analyzing with the SonarQube Scanner

Global Configuration

This step is mandatory if you want to trigger any of your SonarQube analyses with the SonarQube Scanner. You can define as many scanner instances as you wish. Then for each Jenkins job, you will be able to choose with which launcher to use to run the SonarQube analysis.

1. Log into Jenkins as an administrator and go to **Manage Jenkins > Global Tool Configuration**
2. Scroll down to the **SonarQube Scanner** configuration section and click on **Add SonarQube Scanner**. It is based on the typical Jenkins tool auto-installation. You can either choose to point to an already installed version of SonarQube Scanner (uncheck 'Install automatically') or tell Jenkins to grab the installer from a remote location (check '*Install automatically*'):

If you don't see a drop down list with all available SonarQube Scanner versions but instead see an empty text field then this is because Jenkins still hasn't downloaded the required update center file (default period is 1 day). You may force this refresh by clicking 'Check Now' button in **Manage Plugins > Advanced tab**.

Job Configuration

1. **Configure** the project, and scroll down to the **Build** section.
2. Add the *SonarQube Scanner* build step to your build.
3. Configure the SonarQube analysis properties. You can either point to an existing *sonar-project.properties* file or set the analysis properties directly in the ***Analysis properties*** field

Analyzing with SonarQube Scanner for MSBuild

Global Configuration

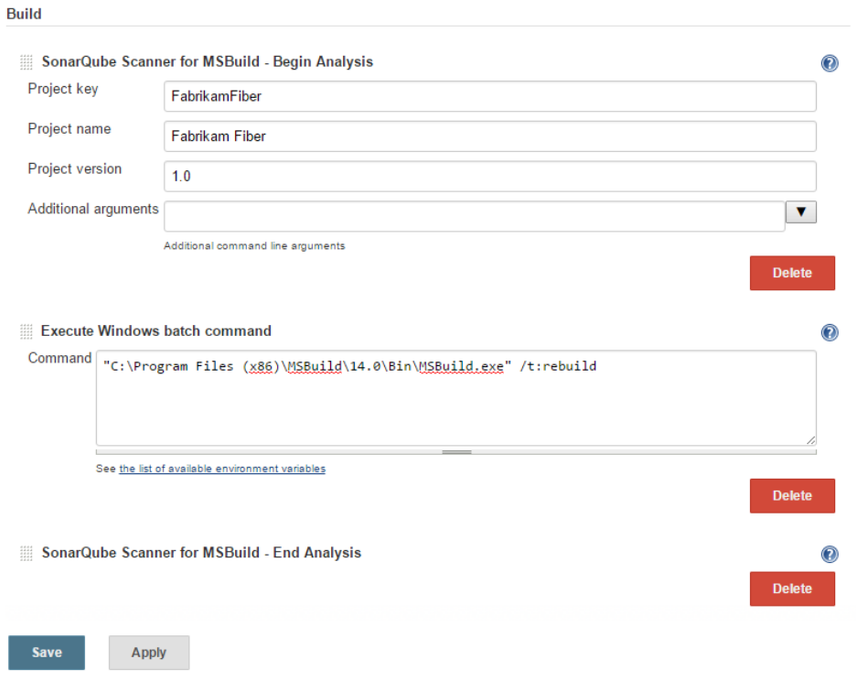
This step is mandatory if you want to trigger any of your analyses with the SonarQube Scanner for MSBuild. You can define as many scanner instances as you wish. Then for each Jenkins job, you will be able to choose with which launcher to use to run the SonarQube analysis.

1. Log into Jenkins as an administrator and go to **Manage Jenkins > Global Tool Configuration**
2. Click on **Add SonarQube Scanner for MSBuild**
3. Add an installation of the latest available version. Check **Install automatically** to have the SonarQube Scanner for MSBuild automatically provisioned on your Jenkins executors

> If you do not see any available version under **Install from GitHub**, first go to **Manage Jenkins** > **Manage Plugins** > **Advanced** and click on **Check now**

Job Configuration

1. **Configure** the project, and scroll down to the **Build** section.
2. Add both the *SonarQube for MSBuild - Begin Analysis* and *SonarQube for MSBuild - End Analysis* build steps to your build
3. Configure the SonarQube **Project Key**, **Name** and **Version** in the **SonarQube Scanner for MSBuild - Begin Analysis** build step
4. Use the **MSBuild** build step or the **Execute Windows batch command** to execute the build with MSBuild 14 (see compatibility) between the **Begin Analysis** and **End Analysis** steps.



Analyzing with SonarQube Scanner for Maven

Global Configuration

1. Log into Jenkins as an administrator and go to **Manage Jenkins > Configure System**
2. Scroll to the **SonarQube servers** section and check *Enable injection of SonarQube server configuration as build environment variables*

Job Configuration

1. **Configure** the project, and scroll down to the **Build Environment** section.
2. Enable *Prepare SonarQube Scanner environment* to allow the injection of SonarQube server values into this particular job. If multiple SonarQube instances are configured, you will be able to choose which one to use.

> Press the ***help*** button to learn which variables you can use in your build. Some values may be blank, depending on what was defined for the server.
>
> 
>

3. Once the environment variables are available, use them in a standard Maven build step by setting the Goals to include

```
$SONAR_MAVEN_GOAL -Dsonar.host.url=$SONAR_HOST_URL
```

⚠ The Post-build Action for Maven analysis is still available, but is deprecated.

Analyzing in a Jenkins pipeline

Since **version 2.5** of the SonarQube Scanner for Jenkins, there is an official support of Jenkins pipeline. We provide a '`withSonarQubeEnv`' block that allow to select the SonarQube server you want to interact with. Connection details you have configured in Jenkins global configuration will be automatically passed to the scanner.

⚠ Support of pipeline only works with **SonarQube >= 5.2**.

Here are a some examples for every scanner, assuming you run on Unix slaves and you have configured a server named 'My SonarQube Server' as well as required tools. If you run on Windows slaves, just replace 'sh' by 'bat'.

SonarQube Scanner

```
node {
  stage('SCM') {
    git 'https://github.com/foo/bar.git'
  }
  stage('SonarQube analysis') {
    // requires SonarQube Scanner 2.8+
    def scannerHome = tool 'SonarQube Scanner 2.8';
    withSonarQubeEnv('My SonarQube Server') {
      sh "${scannerHome}/bin/sonar-scanner"
    }
  }
}
```

SonarQube Scanner for Gradle

```
node {
  stage('SCM') {
    git 'https://github.com/foo/bar.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('My SonarQube Server') {
      // requires SonarQube Scanner for Gradle 2.1+
      // It's important to add --info because of SONARJNKNS-281
      sh './gradlew --info sonarqube'
    }
  }
}
```

SonarQube Scanner for Maven

```
node {
  stage('SCM') {
    git 'https://github.com/foo/bar.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('My SonarQube Server') {
      // requires SonarQube Scanner for Maven 3.2+
      sh 'mvn org.sonarsource.scanner.maven:sonar-maven-plugin:3.2:sonar'
    }
  }
}
```

SonarQube Scanner for MSBuild

```
node {
  stage('SCM') {
    git 'https://github.com/foo/bar.git'
  }
  stage('Build + SonarQube analysis') {
    def sqScannerMsBuildHome = tool 'Scanner for MSBuild 2.2'
    withSonarQubeEnv('My SonarQube Server') {
      // Due to SONARMSBRU-307 value of sonar.host.url and credentials should be passed on command
      bat "${sqScannerMsBuildHome}\\SonarQube.Scanner.MSBuild.exe begin /k:myKey /n:myName /v:1.0
      bat 'MSBuild.exe /t:Rebuild'
      bat "${sqScannerMsBuildHome}\\SonarQube.Scanner.MSBuild.exe end"
    }
  }
}
```

Pause pipeline until quality gate is computed

The `waitForQualityGate` step will pause the pipeline until SonarQube analysis is completed and returns quality gate status.

Pre-requisites:

- SonarQube server 6.2+ (need webhook feature)
- Configure a webhook in your SonarQube server pointing to `<your Jenkins instance>/sonarqube-webhook/` The trailing slash is mandatory with SonarQube 6.2 and 6.3!
- Use `withSonarQubeEnv` step in your pipeline (so that SonarQube taskId is correctly attached to the pipeline context).

Example:

SonarQube Scanner for MSBuild

```
node {
  stage('SCM') {
    git 'https://github.com/foo/bar.git'
  }
  stage('SonarQube analysis') {
    withSonarQubeEnv('My SonarQube Server') {
      sh 'mvn clean package sonar:sonar'
    } // SonarQube taskId is automatically attached to the pipeline context
  }
}

// No need to occupy a node
stage("Quality Gate"){
  timeout(time: 1, unit: 'HOURS') { // Just in case something goes wrong, pipeline will be killed
    def qg = waitForQualityGate() // Reuse taskId previously collected by withSonarQubeEnv
    if (qg.status != 'OK') {
      error "Pipeline aborted due to quality gate failure: ${qg.status}"
    }
  }
}
```

Thanks to the webhook, the step is implemented in a very lightweight way: no need to occupy a node, and it doesn't prevent Jenkins to restart (step will be restored after restart).