[SonarQube Scanner plugin](https://plugins.jenkins.io/sonar/" \t "_blank) version 2.11 or later is required.

1. From the Jenkins Dashboard, navigate to **Manage Jenkins > Manage Plugins** and install the **SonarQube Scanner** plugin.
2. Back at the Jenkins Dashboard, navigate to **Credentials > System** from the left navigation.
3. Click the **Global credentials (unrestricted)** link in the **System** table.
4. Click **Add credentials** in the left navigation and add the following information:
   * **Kind**: Secret Text
   * **Scope**: Global
   * **Secret**: Generate a token at **User > My Account > Security** in SonarQube, and copy and paste it here.
5. Click **OK**.
6. From the Jenkins Dashboard, navigate to **Manage Jenkins > Configure System**.
7. From the **SonarQube Servers** section, click **Add SonarQube**. Add the following information:
   * **Name**: Give a unique name to your SonarQube instance.
   * **Server URL**: Your SonarQube instance URL.
   * **Credentials**: Select the credentials created during step 4.
8. Click **Save**

[GitHub](https://docs.sonarqube.org/latest/analysis/jenkins/)

[GitHub Branch Source plugin](https://plugins.jenkins.io/github-branch-source/) version 2.7.1 or later is required

1. From the Jenkins Dashboard, navigate to **Manage Jenkins > Manage Plugins** and install the **GitHub Branch Source** plugin.
2. From the Jenkins Dashboard, navigate to **Manage Jenkins > Configure System**.
3. From the **GitHub** or **GitHub Enterprise Servers** section, add your GitHub server.
4. Click **Save**.

[GitLab](https://docs.sonarqube.org/latest/analysis/jenkins/)

[GitLab Branch Source plugin](https://plugins.jenkins.io/gitlab-branch-source/) version 1.5.3 or later is required

1. From the Jenkins Dashboard, navigate to **Manage Jenkins > Manage Plugins** and install the **GitLab Branch Source** plugin.
2. From the Jenkins Dashboard, navigate to **Manage Jenkins > Configure System**.
3. From the **GitLab** section, add your GitLab server. Make sure to check the **Manage Web Hooks** checkbox.
4. Click **Save**.

SonarScanner:

node {

stage('SCM') {

git 'https://github.com/foo/bar.git'

}

stage('SonarQube analysis') {

def scannerHome = tool 'SonarScanner 4.0';

withSonarQubeEnv('My SonarQube Server') { // If you have configured more than one global server connection, you can specify its name

sh "${scannerHome}/bin/sonar-scanner"

}

}

}

SonarScanner for Gradle:

node {

stage('SCM') {

git 'https://github.com/foo/bar.git'

}

stage('SonarQube analysis') {

withSonarQubeEnv() { // Will pick the global server connection you have configured

sh './gradlew sonarqube'

}

}

}

SonarScanner for Maven:

node {

stage('SCM') {

git 'https://github.com/foo/bar.git'

}

stage('SonarQube analysis') {

withSonarQubeEnv(credentialsId: 'f225455e-ea59-40fa-8af7-08176e86507a', installationName: 'My SonarQube Server') { // You can override the credential to be used

sh 'mvn org.sonarsource.scanner.maven:sonar-maven-plugin:3.7.0.1746:sonar'

}

}

}

SonarScanner for .NET:

node {

stage('SCM') {

git 'https://github.com/foo/bar.git'

}

stage('Build + SonarQube analysis') {

def sqScannerMsBuildHome = tool 'Scanner for MSBuild 4.6'

withSonarQubeEnv('My SonarQube Server') {

bat "${sqScannerMsBuildHome}\\SonarQube.Scanner.MSBuild.exe begin /k:myKey"

bat 'MSBuild.exe /t:Rebuild'

bat "${sqScannerMsBuildHome}\\SonarQube.Scanner.MSBuild.exe end"

}

}

}

**Pause pipeline until the Quality Gate is computed**

The waitForQualityGate step will pause the pipeline until SonarQube analysis is completed and returns Quality Gate status.

**Pre-requisites:**

* Configure a webhook in your SonarQube server pointing to <your Jenkins instance>/sonarqube-webhook/
* Use withSonarQubeEnv step in your pipeline (so that SonarQube task-Id is correctly attached to the pipeline context).

Scripted pipeline example:

node {

stage('SCM') {

git 'https://github.com/foo/bar.git'

}

stage('SonarQube analysis') {

withSonarQubeEnv('My SonarQube Server') {

sh 'mvn clean package sonar:sonar'

} // submitted 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 after a timeout

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 doing polling, and it doesn't prevent Jenkins to restart (step will be restored after restart). Note that to prevent race conditions, when the step starts (or is restarted) a direct call is made to the server to check if the task is already completed.

Declarative pipeline example:

pipeline {

agent any

stages {

stage('SCM') {

steps {

git url: 'https://github.com/foo/bar.git'

}

}

stage('build && SonarQube analysis') {

steps {

withSonarQubeEnv('My SonarQube Server') {

// Optionally use a Maven environment you've configured already

withMaven(maven:'Maven 3.5') {

sh 'mvn clean package sonar:sonar'

}

}

}

}

stage("Quality Gate") {

steps {

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

// Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

// true = set pipeline to UNSTABLE, false = don't

waitForQualityGate abortPipeline: true

}

}

}

}

}

If you want to run multiple analysis in the same pipeline and use waitForQualityGate you have to do everything in order:

pipeline {

agent any

stages {

stage('SonarQube analysis 1') {

steps {

sh 'mvn clean package sonar:sonar'

}

}

stage("Quality Gate 1") {

steps {

waitForQualityGate abortPipeline: true

}

}

stage('SonarQube analysis 2') {

steps {

sh 'gradle sonarqube'

}

}

stage("Quality Gate 2") {

steps {

waitForQualityGate abortPipeline: true

}

}

}

}

**Configuring a webhook secret**

If you want to verify the webhook payload that is sent to Jenkins, you can add a secret to your webhook on SonarQube.

To set the secret:

1. In Jenkins, navigate to **Manage Jenkins > Configure System > SonarQube Server > Advanced > Webhook Secret** and click the **Add** button.
2. Select **Secret text** and give the secret an ID.
3. Select the secret from the dropdown menu.

If you want to override the webhook secret on a project level, you can add the secret to Jenkins and then reference the secret ID when calling waitForQualityGate.

waitForQualityGate(webhookSecretId: 'yourSecretID')