

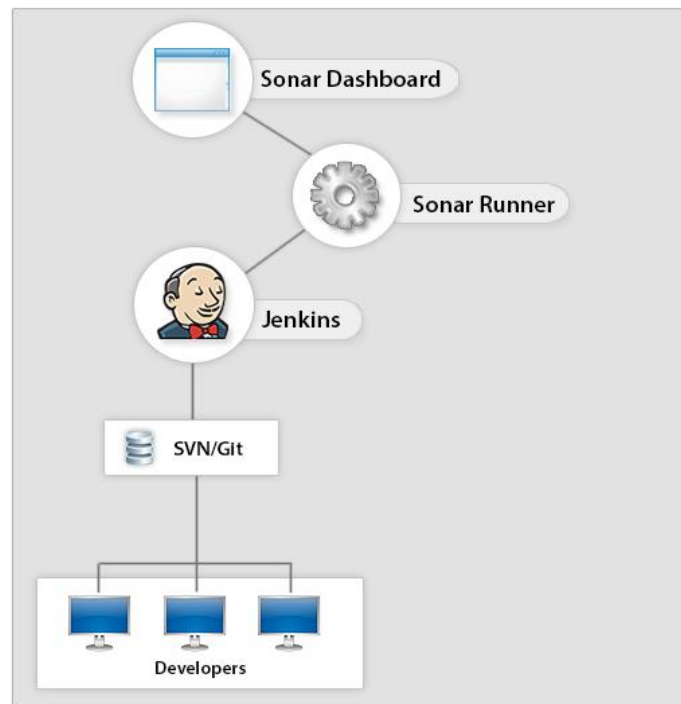
0. Continuous integration and static code analysis

Continuous integration deals with merging code implemented by multiple developers into a single build system. Developers frequently integrate their code and the final build is automated, developer unit test are executed automatically to ensure the stability of the build. This approach is inspired by extreme programming methodologies. With a test driven approach put into place continuous integration would yield in the following benefits.

- + Identifying bugs and issues immediately as they arise.
- + Automated deployment and configuration to minimize manual deployment time.
- + Supports in generating documentation (Java Docs) automatically as post build steps.
- + Identifying conflicts in changes integrated.
- + Easier revert back to stable builds.
- + Metrics which provide code complexity and coverage.

Static code analysis analyzes source code for common coding standards and guidelines and notifies common code smells. Static code analysis performs analysis on uncompiled, unexecuted code. Even though there are extensions and tools for IDEs that allow individual developers to perform static code analysis, its important to note that integrating with continuous integration tools allows a more consistent and uniform analysis for all developers.

In this document we would be taking a closer look at setting up a **Jenkins** server, which would be the continuous integration tool, with **SonarQube** a static code analysis tool. The figure below illustrates how all of the tools would fit into place.



Individual developers would commit the code to a code repository. Jenkins would make checkouts of the code from the repository and would perform automated builds and would execute unit tests. During this process it would run a sonarqube runner which ultimately integrates the static analysis results to the SonarQube dashboard.

The SonarQube dashboard and the Jenkins server can be deployed as two separate components. The SonarQube Jenkins plugin ultimately executes the sonar runner to integrate the code to the sonar dashboard. This automated process can be repetitively performed at scheduled intervals.

1. Install, Confiure Jenkins CI

Update system

```
$ sudo apt-get update
```

Install Apache web server

```
$ sudo apt-get install apache2
```

Install Oracle JDK 7

```
$ sudo apt-get purge openjdk*
```

```
$ sudo apt-get install python-software-properties
```

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

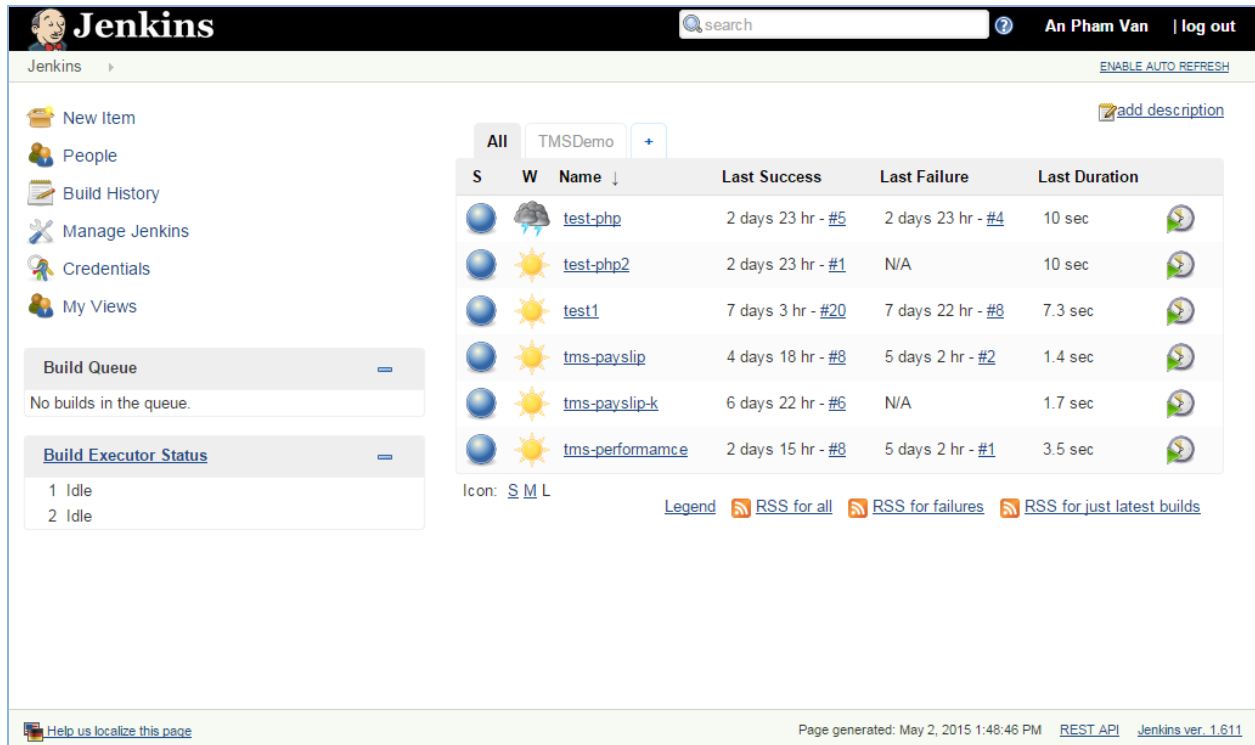
Install Jenkins CI

```
$ wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
$ sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt-get update
$ sudo apt-get install jenkins
```

Upgrade Jenkins CI

Once installed like this, you can update to the later version of Jenkins (when it comes out) by running the following commands:

```
$ sudo apt-get update
$ sudo apt-get install jenkins
```



The screenshot shows the Jenkins CI web interface. The top navigation bar includes the Jenkins logo, a search bar, and the user name 'An Pham Van' with a 'log out' link. The left sidebar contains links for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'Credentials', and 'My Views'. The main content area displays a table of builds for the 'TMSDemo' job. The table has columns for status (S), weather icon (W), name, last success, last failure, and last duration. Below the table, there are links for 'Icon: S M L' and 'Legend' with RSS feeds for all, failures, and latest builds. The bottom of the page shows a footer with 'Page generated: May 2, 2015 1:48:46 PM', 'REST API', and 'Jenkins ver. 1.611'.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		test-php	2 days 23 hr - #5	2 days 23 hr - #4	10 sec
		test-php2	2 days 23 hr - #1	N/A	10 sec
		test1	7 days 3 hr - #20	7 days 22 hr - #8	7.3 sec
		tms-payslip	4 days 18 hr - #8	5 days 2 hr - #2	1.4 sec
		tms-payslip-k	6 days 22 hr - #6	N/A	1.7 sec
		tms-performance	2 days 15 hr - #8	5 days 2 hr - #1	3.5 sec

2. Configure Jenkins CI to work with GitLab

2.1. Source Code Management - Git Configuration

* In GitLab:

Jenkins need read access to GitLab repository. We already specified private key to use in Jenkins. Now we need to add public key to GitLab project

*** In Jenkins CI:**

- + Install GitLab Hook plugin
- + Setup jenkins project

In the Branch Specifier field, enter your desired branch, depending on which one you're going to build in this current job. For master simply enter */master, for building your feature/userstory branches (and given you start them with story/<name>), enter */story/* and so on.

2.2. GitLab Web Hook

The next step is to add a web hook to GitLab. This is needed s.t. GitLab is able to signal to Jenkins about new commits that have been "pushed" to the repository. This is an alternative, but much more efficient way of instead doing a continuous polling.

Just go to your repository settings and then to Web Hooks, enter an url of the form
http://your-jenkins-server/git/notifyCommit?url=<URL of the Git repository for the Gitlab project>

3. Install SonarQube, SonarQube Runner

3.1. Install SonarQube

Add the following entry in your `/etc/apt/sources.list` :

```
deb http://downloads.sourceforge.net/project/sonar-pkg/deb binary/
```

Update your local package index and install Sonar:

```
$ sudo apt-get update
$ sudo apt-get install sonar
```

Install MySQL, login as root and create the necessary tables and permissions (change the password of the MySQL sonar user to something else):

```
$ sudo apt-get install mysql-server
$ mysql -u root -p
```

```
CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'sonar' IDENTIFIED BY 'sonar';
GRANT ALL ON sonar.* TO 'sonar'@'%' IDENTIFIED BY 'sonar';
GRANT ALL ON sonar.* TO 'sonar'@'localhost' IDENTIFIED BY 'sonar';
FLUSH PRIVILEGES;
```

Edit the file at `/opt/sonar/conf/sonar.properties` and set the following (disable default driver):

```
#----- Embedded Database (default)
# It does not accept connections from remote hosts, so the
# server and the analyzers must be executed on the same host.
# sonar.jdbc.url: jdbc:h2:tcp://localhost:9092/sonar
# sonar.jdbc.driverClassName: org.h2.Driver
#----- PostgreSQL 8.x/9.x
# If you don't use the schema named "public", please refer to
http://jira.codehaus.org/browse/SONAR-5000
#sonar.jdbc.url=jdbc:postgresql://localhost/sonar
```

Uncomment the MySQL settings section as follows:

```
#----- MySQL 5.x
# Comment the embedded database and uncomment the following line to use MySQL
sonar.jdbc.url:
jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatched
Statements=true
# Optional properties
sonar.jdbc.driverClassName: com.mysql.jdbc.Driver
```

And make sure the user name and password to connect to the data base are set:

```
# User credentials.
# Permissions to create tables, indices and triggers must be granted to JDBC user.
# The schema must be created first.
sonar.jdbc.username: sonar
sonar.jdbc.password: sonar
```

Start the Sonar server and visit the url (`http://localhost:9000`) to confirm if it works:

```
$ sudo service sonar start
```

sonarqube Dashboards ▾ Issues Measures Rules Quality Profiles Quality Gates Settings More ▾ Administrator ▾ 🔍 ?

Since you are able to read this, it means that you have successfully started your SonarQube server. Well done!

If you have not removed this text, it also means that you have not yet played much with SonarQube. So here are a few pointers for your next step:

- » Do you now want to run analysis on a project?
- » Maybe start customizing dashboards?
- » Or simply browse the [complete documentation](#)?
- » If you have a question or an issue, please visit the [Get Support](#) page.

MY FAVOURITES

QG NAME ▴ LAST ANALYSIS

No data

PROJECTS

Size: Lines of code Color: Coverage

PHP project analyzed with the SonarQube Runner

PHP project analyzed with the SonarQube Runner reusing PHPUnit reports

2 results

SonarQube™ technology is powered by SonarSource SA
Version 5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - Web Service API

sonarqube Dashboards ▾ Issues Measures Rules Quality Profiles Quality Gates Settings More ▾ Administrator ▾ 🔍 ?

★ PHP project analyzed with the SonarQube Runner reusing PHPUnit reports Version 1.0 / April 29 2015 2:24 PM

Overview Components Issues Settings ▾ More ▾

Main Dashboard Configure widgets

Lines Of Code <u>226</u> PHP	Files <u>2</u> Directories <u>3</u> Lines <u>426</u>	SQALE Rating <u>A</u>	Technical Debt Ratio <u>1.9%</u>
Functions <u>9</u> Classes <u>2</u> Statements <u>129</u>	Duplications <u>72.8%</u> Lines <u>310</u> Blocks <u>2</u> Files <u>2</u>	Debt <u>2h 10min</u> Issues <u>6</u>	Issues <u>6</u>
Complexity <u>49</u>	Directory Tangle Index <u>0.0%</u> Cycles <u>> 0</u>	Dependencies To Cut Between Directories <u>0</u> Between Files <u>0</u>	

1 Blocker 0
 1 Critical 0
 1 Major 6
 1 Minor 0
 1 Info 0

3.2. Install SonarQube Runner

Download SonarQube Runner.

```
$ wget http://repo1.maven.org/maven2/org/codehaus/sonar/runner/sonar-runner-dist/2.4/sonar-runner-dist-2.4.zip
$ unzip sonar-runner-dist-2.4.zip
```

Edit SonarQube Runner properties.

```
$ vi sonar-runner-2.4/conf/sonar-runner.properties
#Configure here general information about the environment, such as SonarQube DB
details for example
#No information about specific project should appear here

#----- Default SonarQube server
sonar.host.url=http://localhost:9000
```

```
#----- PostgreSQL
#sonar.jdbc.url=jdbc:postgresql://localhost/sonar

#----- MySQL
sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8

#----- Oracle
#sonar.jdbc.url=jdbc:oracle:thin:@localhost/XE

#----- Microsoft SQLServer
#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMethod=Cursor

#----- Global database settings
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar

#----- Default source code encoding
#sonar.sourceEncoding=UTF-8

#----- Security (when 'sonar.forceAuthentication' is set to 'true')
sonar.login=admin
sonar.password=admin
```

Relocate SonarQube Runner.

```
$ sudo mv sonar-runner-2.4/ /usr/local/
$ sudo ln -s /usr/local/sonar-runner-2.4/ /usr/local/sonar-runner
```

Add SonarQube Runner bin to path. Add SonarQube Runner home to environment.

```
$ sudo vi /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/local/sonar-runner/bin:"
SONAR_RUNNER_HOME=/usr/local/sonar-runner
```

4. Integrate SonarQube with Jenkins CI

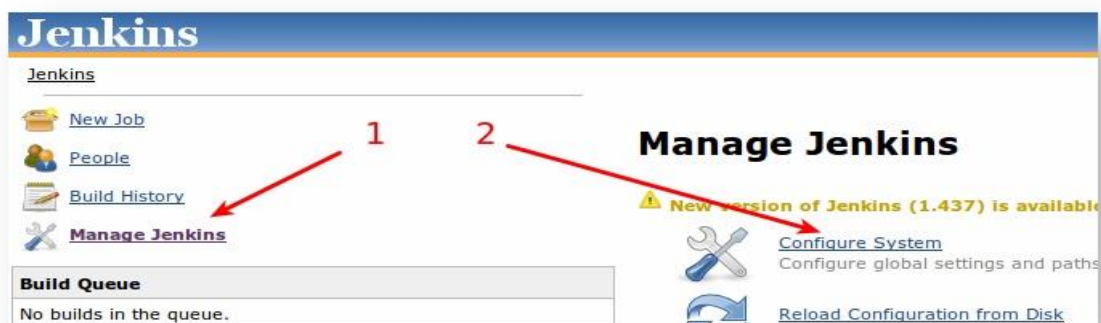
4.1. Configure the SonarQube Jenkins Plugin

4.1.1. Adding SonarQube Server

You can define as many SonarQube servers as you wish. Then for each Jenkins job, you will be able to choose which server to use for the SonarQube analysis.

To add a SonarQube server, just follow the three steps below:

1. Log into Jenkins as an administrator and go to Manage Jenkins > Configure System:



2. Scroll down to the *SonarQube* configuration section and click on *Add SonarQube*:

SonarQube

SonarQube installations

Name

SonarQube Local 5.1-RC1

Disable

☐

Check to quickly disable SonarQube on all jobs.

Advanced...

Delete SonarQube

Name

SonarQube Prod 4.5.4

Disable

☐

Check to quickly disable SonarQube on all jobs.

Advanced...

Delete SonarQube

Add SonarQube

List of SonarQube installations

3. Configure your SonarQube installation by clicking on the "Advanced..." button :

SonarQube installations

Name

SonarQube Local 5.1-RC1

Disable

☐

Check to quickly disable SonarQube on all jobs.

Server URL

SonarQube account login

Default is http://localhost:9000

SonarQube account password

SonarQube account used to perform analysis. Mandatory when anonymous access is disabled.

Database URL

SonarQube account used to perform analysis. Mandatory when anonymous access is disabled.

Database login

Do not set if default embedded database.

Database password

Default is sonar.

Version of sonar-maven-plugin

Default is sonar.

Additional properties

If not specified, then sonar:sonar will be used.

Trigger Exclusions (only for jobs with SonarQube as a post build action)

Skip if triggered by SCM Changes

☐

Skip if triggered by the build of a dependency

☐

Skip if environment variable is defined and set to true

Delete SonarQube

4.1.2. Adding SonarQube Runner

This step is mandatory if you want to trigger any of your SonarQube analyses with the SonarQube Runner. Skip this step if you want to trigger all your analyses with Maven.

You can define as many SonarQube Runner launchers as you wish. Then for each Jenkins job, you will be able to choose with which launcher to use to run the SonarQube analysis.

To add a SonarQube Runner, just follow the three steps below:

1. Log into Jenkins as an administrator and go to Manage Jenkins > Configure System:



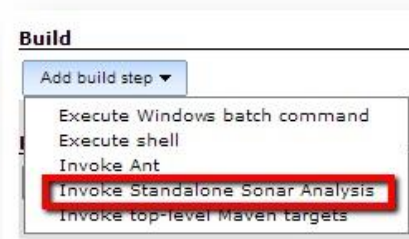
2. Scroll down to the *SonarQube Runner* configuration section and click on *Add SonarQube Runner*. It is based on the typical Jenkins tool auto-installation. You can either choose to point to an already installed version of SonarQube Runner (uncheck 'Install automatically') or tell Jenkins to grab the installer from a remote location (check 'Install automatically'):

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

4.2. Triggering SonarQube on Jenkins Job

4.2.1. Triggering a Project Analysis with the SonarQube Runner

Go to the *Build* section, click on *Add build step* and choose *Invoke Standalone SonarQube Analysis*:



Configure the SonarQube analysis. You can either point to an existing *sonar-project.properties* file or set the analysis properties directly in the *Project properties* field:

Build

Invoke Standalone Sonar Analysis

Sonar Installation: Sonar Local 3.3

JDK: (Inherit From Job)

Sonar Runner: Sonar Runner 2.0

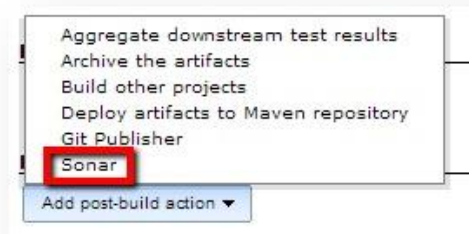
Path to project properties:

Project properties:

JVM Options:

4.2.2. Triggering a Project Analysis with Maven

On a Maven job, go to the 'Post-build Actions' section and click on 'Add post-build action':



Then configure this post-build action:

Choose the *SonarQube Installation*:

Post-build Actions

Sonar

Sonar Installation: Sonar Local 3.3

Advanced...

Delete

- Fill in the 'Advanced...' parameters as needed:

Post-build Actions

Sonar

Sonar Installation: Sonar Local 3.3

Branch:

Language:

JDK: (Inherit From Job)

MAVEN_OPTS:

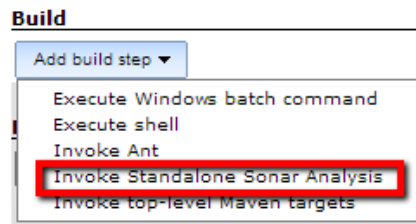
Additional properties:

☐ Dont use global triggers configuration

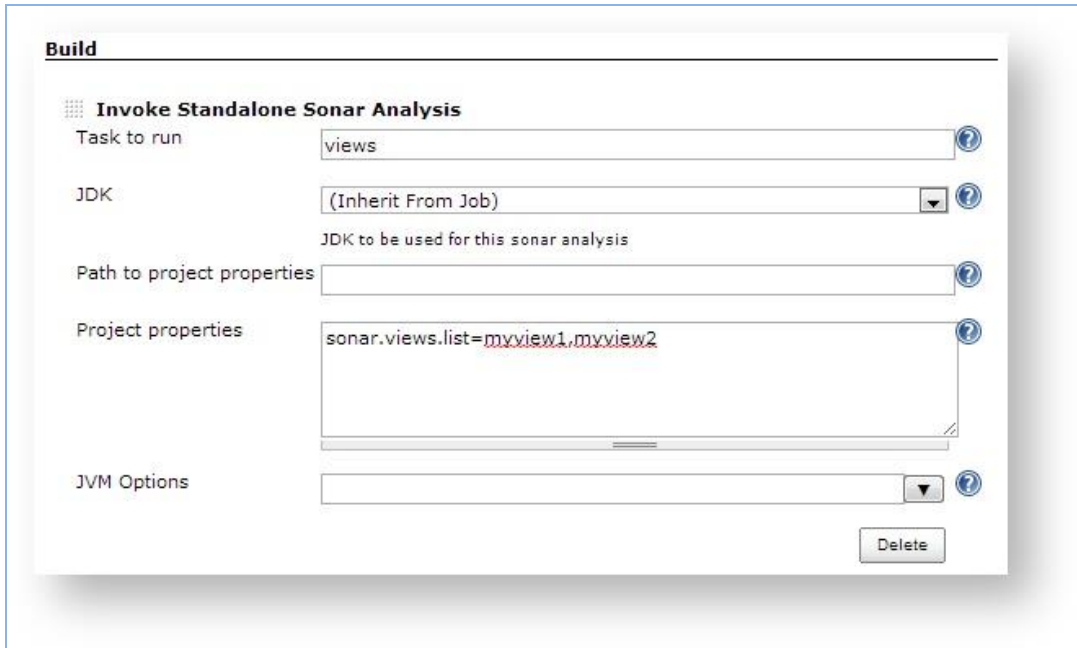
Delete

4.2.3. Triggering a Task with the SonarQube Runner

Go to the *Build* section, click on *Add build step* and choose *Invoke Standalone SonarQube Analysis*:



Configure the task (computation of [views](#), computation of [developers](#), or generation of [reports](#)). If you have more than one task to perform, you'll need to configure multiple build steps:

A screenshot of the 'Invoke Standalone Sonar Analysis' configuration form. The form has a title 'Invoke Standalone Sonar Analysis' and several fields: 'Task to run' with the value 'views', 'JDK' with the value '(Inherit From Job)', 'Path to project properties' (empty), 'Project properties' with the value 'sonar.views.list=myview1,myview2', and 'JVM Options' (empty). There are help icons (question marks) next to the 'Task to run', 'JDK', 'Path to project properties', 'Project properties', and 'JVM Options' fields. A 'Delete' button is located at the bottom right of the form.