**Jenkins**

**1. Using the Jenkins build server**

*Continuous integration* is a process in which all development work is integrated as early as possible and the resulting artifacts are automatically created and tested. This process should identify errors as very early in the process.

*Jenkins* is one open source tool to perform continuous integration and build automation. The basic functionality of Jenkins is to execute a predefined list of steps. The trigger for this execution can be time or event based. For example every 20 minutes or after a new commit in a Git repository.

The list of steps can, for example, include:

* perform a software build with Apache Maven or Gradle
* Run a shell script
* Archive the build result
* Afterwards start the integration tests

Jenkins also monitors the execution of the steps and allows to stop the process if one of the steps fails. Jenkins can also send out notifications about the build success or failure.

*Jenkins* can be extended by additional plug-ins, e.g., for building and testing Android applications or to support the Git version control system.

**2. Installation**

Jenkins can be started via the command line or can run in a web application server. Under Linux you can also install Jenkins as a system service.

For most platforms you have native packages, see the [**Jenkins Homepage**](http://jenkins-ci.org/).

**2.2. Installing of the Jenkins server on Ubuntu**

Jenkins provides Debian/Ubuntu packages which install Jenkins and register Jenkins as start service. See the[**Install Jenkins on Ubuntu description**](https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Ubuntu).

Jenkins stores all the settings, logs and build artifacts in its home directory. The default installation directory is*/var/lib/jenkins* under Ubuntu.

This creates a */etc/init.d/jenkins* start script which starts Jenkins automatically at boot time. If you installed Jenkins locally, you find it running under the following URL: [**http://localhost:8080/**](http://localhost:8080/)

**2.3. Using the .WAR file of Jenkins**

Download the *jenkins.war* file from [**Jenkins Homepage**](http://jenkins-ci.org/).

You can also start Jenkins directly via the command line with java -jar jenkins\*.war. If you start it locally, you find it running under the following URL: [**http://localhost:8080/**](http://localhost:8080/)

To run it in your Tomcat server, put the .WAR file into the *webapps* directory. If you start Tomcat, your Jenkins installation will be available under [**http://localhost:8080/jenkins**](http://localhost:8080/jenkins)

**Note**

If the jenkins.war is deployed in your *webapps* directory, but cannot be started and the tomcat manager says “﻿FAIL - Application at context path /jenkins could not be started ”, you may need to grant the permissons for *﻿JENKINS\_HOME*.

﻿cd /usr/share/tomcat7

﻿sudo mkdir .jenkins

﻿sudo

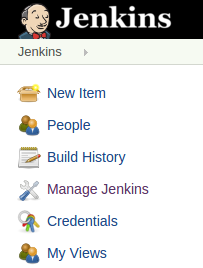
chown tomcat7:nogroup .jenkins

This makes the .jenkins folder writable and Jenkins can use it.

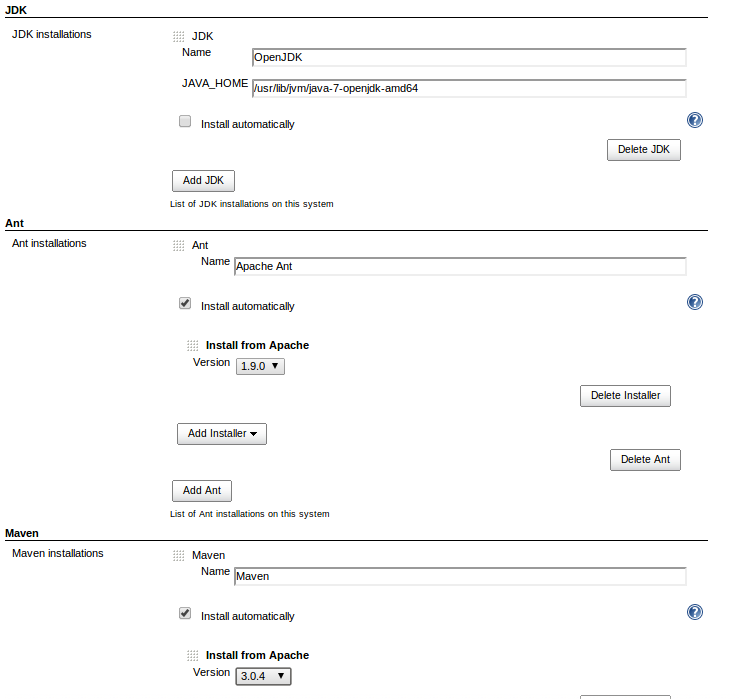
**3. Configure Jenkins**

**3.1. Configuration the JDK location**

Before using Jenkins to build Java applications, you need to configure the location or it where your JDK installation is. Select *Manage Jenkins* and afterwards *Configure System*.

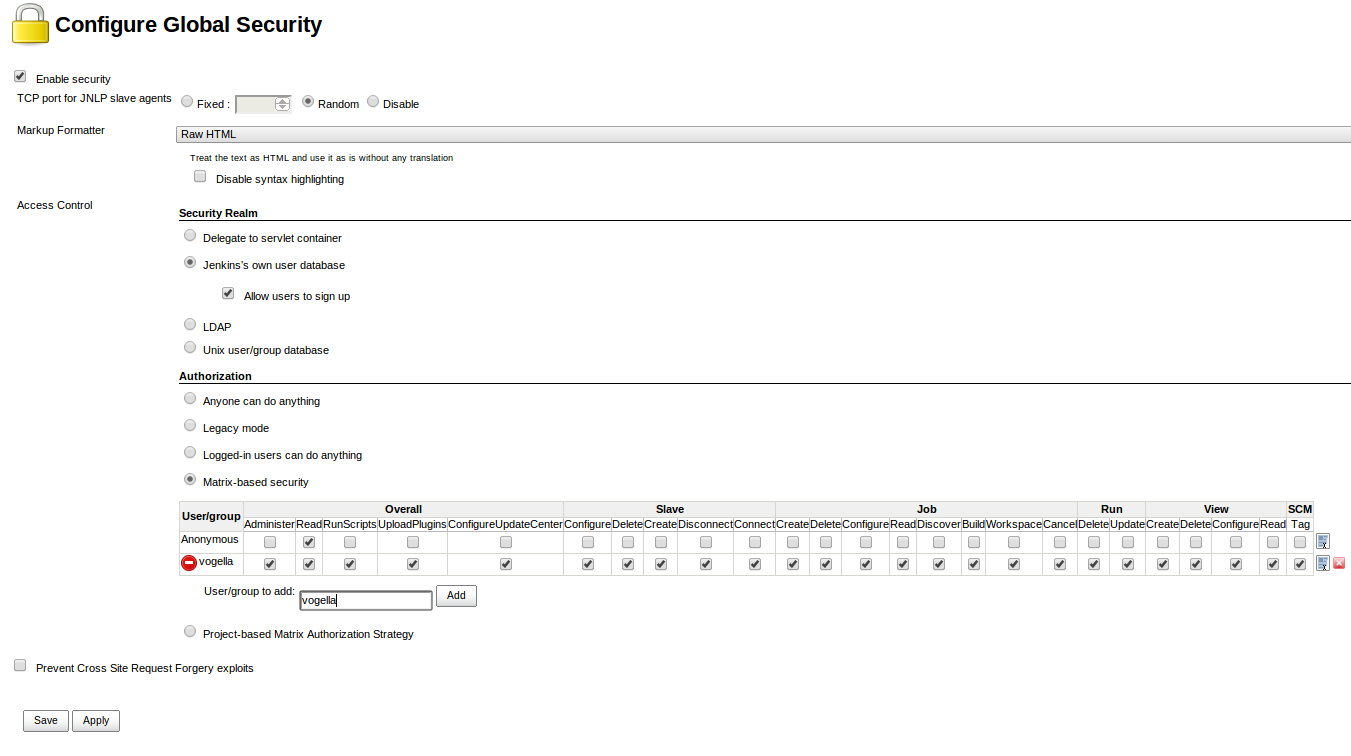


Enter the correct path to your JDK, Apache Ant and Maven and press the *Save* button below. Jenkins can also install these for your automatically.

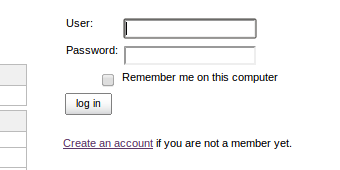


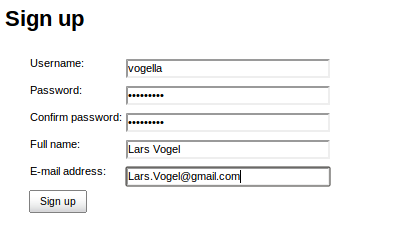
**3.2. Secure Jenkins**

It is recommended to secure Jenkins. *Manage Jenkins* and then *Configure Global Security*. Select the *Enable security* flag. The easiest way is to use Jenkins own user database. Create at least the user "Anonymous" with read access. Also create entries for the users you want to add in the next step.

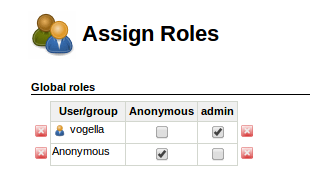


On the login page, select *Create an account* to create the users you just gave access.

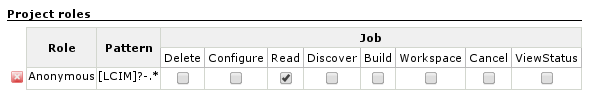




Go to *Manage Jenkins*, *Manage and Assign Roles* and then *Assign Roles* to grant the newly created user additional access rights.



Navigate to *Manage Roles* to define access restrictions in detail. *Pattern* is a regex value of the job name. The following grants unregistered users read-only access to your build jobs that start with the C-MASTER or M-MASTERprefix and only those.



**3.3. Generate ssh key for Jenkins user**

If you want to access a private Git repo, for example at Github, you need to generate an ssh key-pair. Create a SSH key with the following command: sudo -u jenkins ssh-keygen.

**4. Jenkins management**

**4.1. Plug-in management**

Jenkins can be extended via additional plug-ins with more functionality. You can configure your plug-ins via the*Manage Jenkins* → *Manager Plugins* link.

To install plugins in Jenkins select use the *Manage Jenkins* → *Manager Plugins* link and search for the plugin you want to install. Select it from the list and select to install it and restart Jenkins.

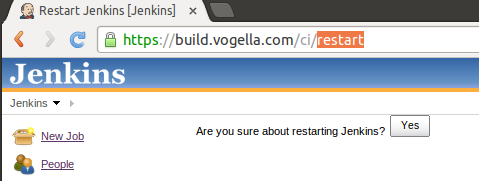
The following table is a summary of commonly used plug-ins.

**Table 1. Jenkins plug-ins**

| **Plug-in name** | **Description** | **URL** |
| --- | --- | --- |
| Git Plugin | This plugin allows use of Git as a build SCM. | https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin |
| Xvnc plugin | This plugin allows projects to run xvnc during a build. This allows for example to run tests which requires a display to run on a virtual display. To use this plug-in you need to connect once to your vncserver on the command line to provide a password. Use for example the following commands.  # install vncserver  apt-get install vnc4server  # **switch** to jenkins user  sudo su jenkins  # connect to vncserver which creates the password  vncserver :10 | wiki.jenkins-ci.org/display/JENKINS/Xvnc+Plugin |
| Gradle Plugin | This plugin allows to run Gradle builds, e.g., as required for Android, via Jenkins. | https://wiki.jenkins-ci.org/display/JENKINS/Gradle+Plugin |
| Maven Plugin | This plugin allows to run Maven builds. | https://wiki.jenkins-ci.org/display/JENKINS/Maven+Project+Plugin |
| GitHub plugin | This plugin integrates Jenkins with Github projects. | https://wiki.jenkins-ci.org/display/JENKINS/Github+Plugin |
| Publish Over SSH Plugin | This plugin allows to publish build artifacts via ssh | https://wiki.jenkins-ci.org/display/JENKINS/Publish+Over+SSH+Plugin |
| Workspace Cleanup Plugin | This plugin allows to delete the workspace before the build or when a build is finished and artifacts saved. | https://wiki.jenkins-ci.org/display/JENKINS/Workspace+Cleanup+Plugin |
| Github Pull Request Builder | This plugin allows to build Github Pull Requests | https://wiki.jenkins-ci.org/display/JENKINS/GitHub+pull+request+builder+plugin |

**4.2. Restart your Jenkins**

You can manually restart Jenkins by adding *restart* as URL parameter.



**5. Support for the Git version control systems**

Jenkins supports the Git version control system via a plugin. Select the *Manage Jenkins* → *Manager Plugins* link. Here you have to install the *Git Plugin*.

To clone a Git repostory via Jenkins you need to enter the email and user name for your Jenkins system. For this switch into your job directory and run the git config command.

*# Need to configure the Git email and user for the Jenkins job*

*# switch to the job directory*

**cd** /var/lib/jenkins/jobs/Android/workspace

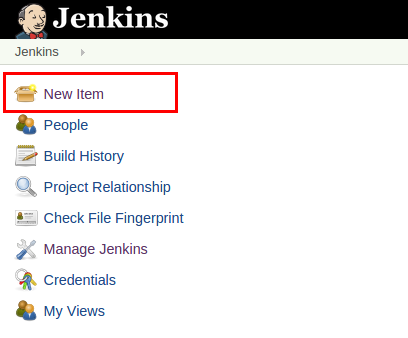
*# setup name and email*

sudo git config user.name "jenkins"

sudo git config user.email "test@gmail.com"

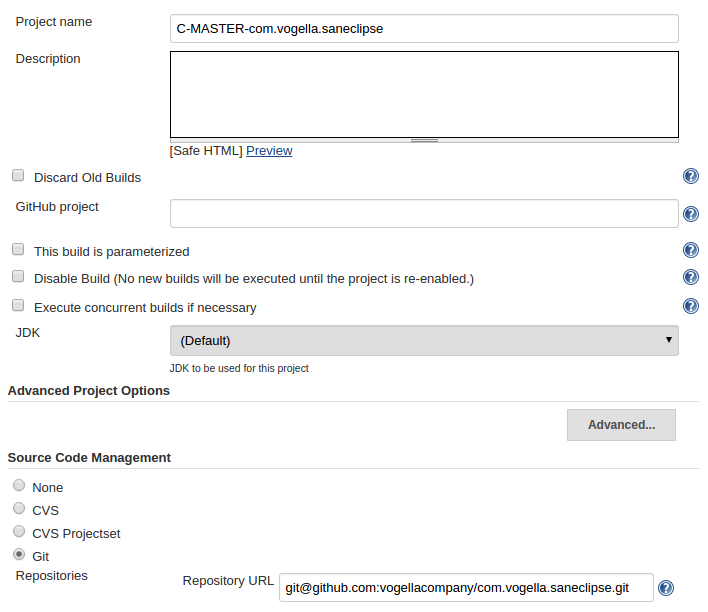
**6. Setting up a Jenkins job**

The build of a project is handled via *jobs* in Jenkins. Select *New Item* from the menu

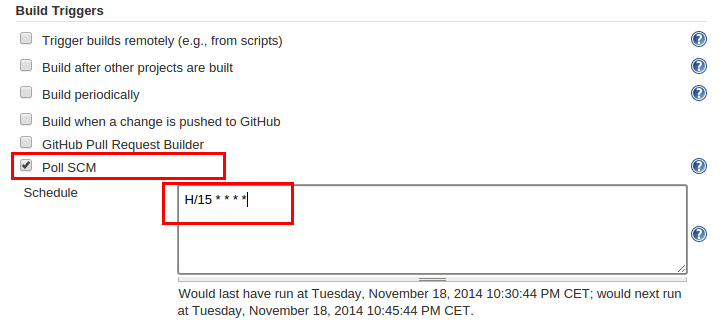


Afterwards enter a name for the job and select *Freestyle Job*. Press *OK* to create a new Job in Jenkins.

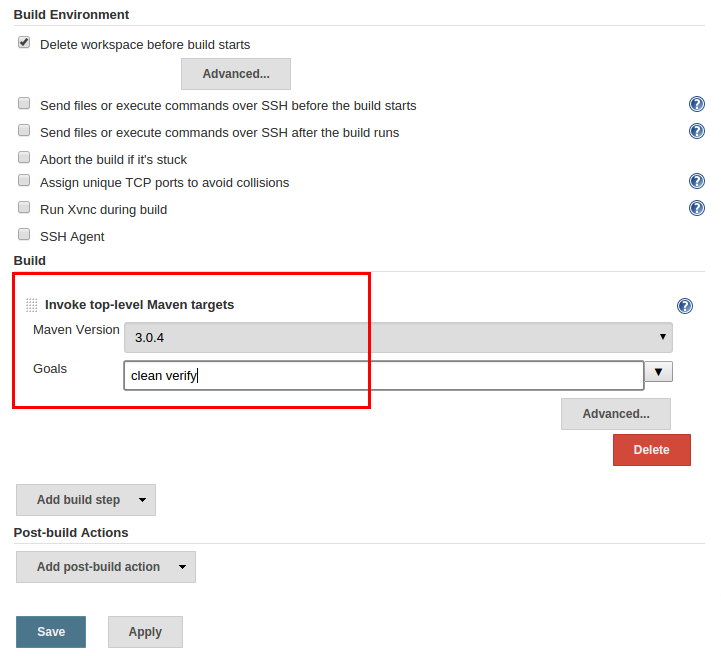
The next page allows you to configure your job. If you for example using Git, enter the URL to the Git repository. If the repository is not public, you may also need to configure the credentials.



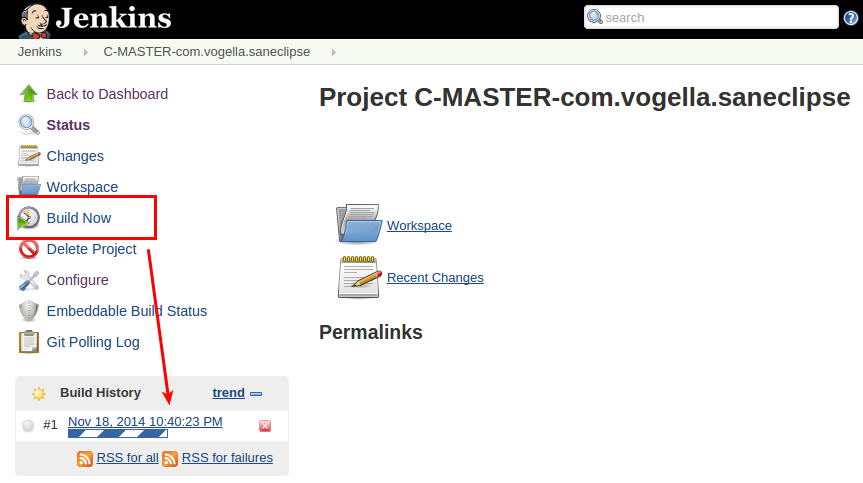
Specify when and how your build should be triggered. The following example polls the Git repository every 15 min and triggers a build, if something has changed in the repo.



I typically delete the workspace before a build to avoid any side-effect. In the Build section you can add a build step, e.g., a Maven build.



Press *Save* to finish the job definition. Press *Build Now*on the job page to validate the job works as expected.



After a while the job should go to green or blue (depending on your configuration), if successful. Click on the job and afterwards on *Console Output* to see the log file in case of an error or to validate that log looks as expected.

