Usage Guide

**Jenkins**

Table of Contents

[1 Introduction 3](#_Toc355027944)

[2 Creating a Job in Jenkins 3](#_Toc355027945)

[3 Build Results and Historic Information 8](#_Toc355027946)

[4 Insights for Changes Between Builds 12](#_Toc355027947)

[5 Dependencies and Artefact Fingerprints 13](#_Toc355027948)

[6 References 16](#_Toc355027949)

[7 Document Control 16](#_Toc355027950)

[7.1 Change History 16](#_Toc355027951)

[7.2 Open Issues 17](#_Toc355027952)

# Introduction

Jenkins is a Continuous Integration engine that enables easy configuration and management of automated builds, release and deployment management and continuous tests/quality assurance.

Jenkins is the recommended tool in Java Blueprint 3 for adopting Continuous Integration practices.

This document contains detailed insights on how to use Jenkins from the end user perspective: creating jobs, how to configure and how to run them. The companion document, **Jenkins Installation and Setup Guide**, includes information for administrators on how to install and configure Jenkins for first use plus some typical administrative tasks.

# Creating a Job in Jenkins

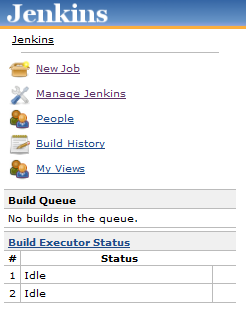
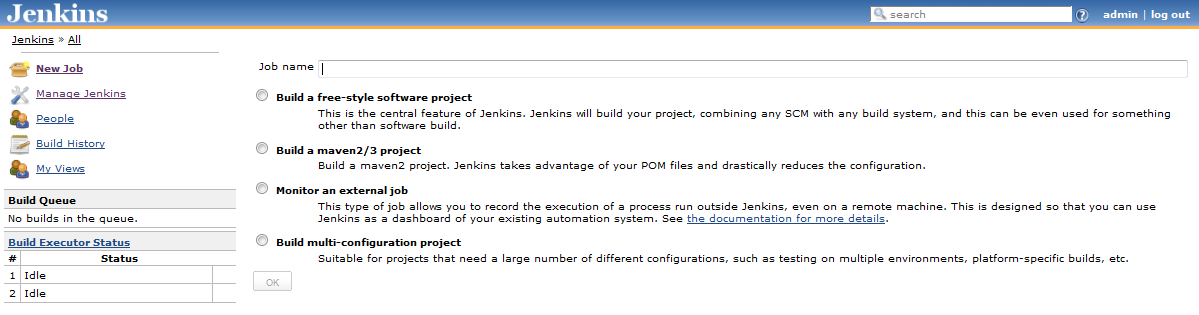
A job in Jenkins is the entity that represents the automation of build, test and quality assurance tasks for a software project, application or module.

In general we will create jobs for every application that we want to control under Continuous Integration (CI) practices. Granularity may vary: a module or a whole application or system. We may even setup multiple jobs to handle multiple branches for the same project.

It is also a common practice to setup jobs with different tasks, triggers or schedules. Typical use case scenarios are:

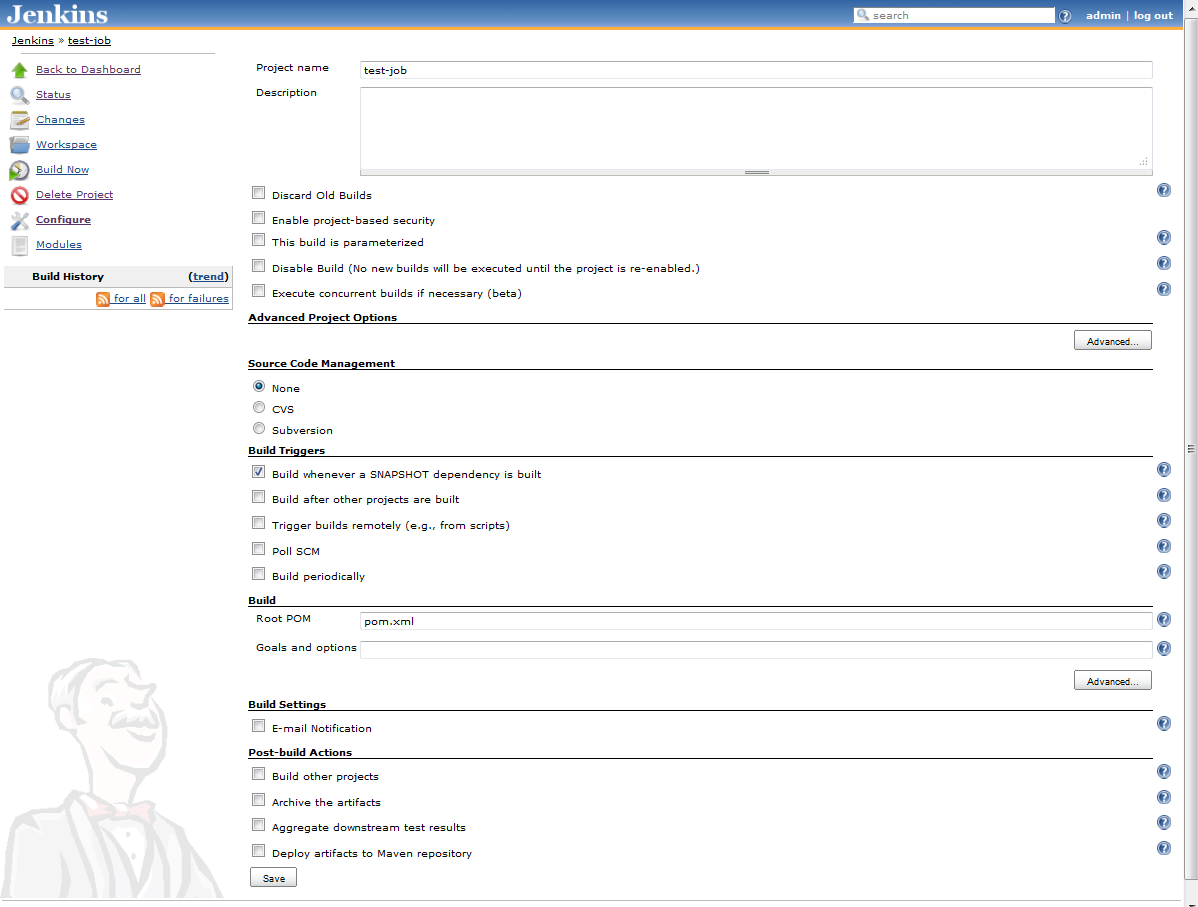
* A continuous job that triggers after every check-in in the source code management system, automating compilation and unit test tasks.
* A job that triggers when a dependant artefact is updated. For example artefact A depends on B, so we should check that A is working fine every time that B is modified.
* A job scheduled daily at nights that downloads a snapshot from the source code management system, compiles all code, run unit tests and checks quality using Java Blueprint recommended tools (either individually or consolidated using Sonar – Continuous Quality Assurance, CQA, approach).

To create a new job in Jenkins, access the dashboard URL and login with administrator credentials. In the left menu, select **New Job**. The next screen will show the different job types that can be created (more may appear depending on plugins installed):

To proceed, add the job name and select the job type. In this guide we will assume that a Maven 2/3 project will be created, as it is the usual job type for Java technology projects. Apache Maven is the Java Blueprint recommended tool for build automation. Maven is capable to automate many of the common tasks needed through the lifecycle of a software artefact – compilation, resource management, unit and assembly tests, packaging and deployment – all of that with minimal configuration. Jenkins leverage Maven allowing easy and quick configuration of jobs.

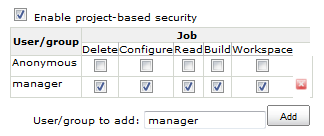
Once you are ready with the selection, press OK to continue. The job configuration screen will appear:



There are several sections in the job configuration screen. Default ones are shown on the screenshot above, but many other sections may appear depending on the plugins that have been installed in Jenkins.

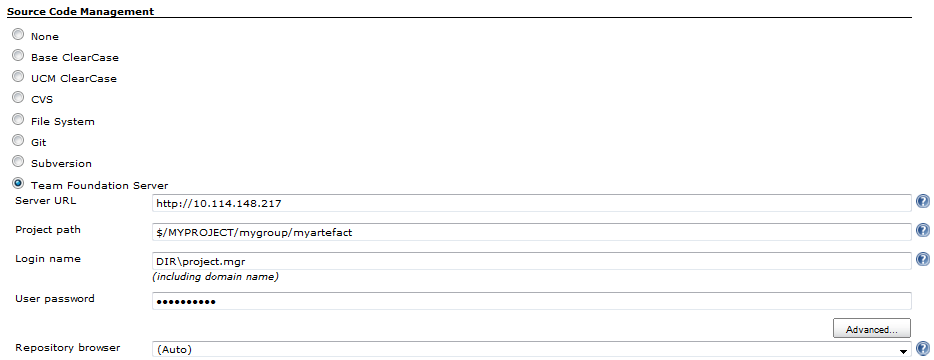
The general data section is where you can configure the project name and description and other general options. The recommended configuration is the following:

* Select the check-box **Discard Old Builds** to reduce the amount of space on disk needed. Two options will appear below the check-box to select the number of builds or the number of days after which build results will be cleaned up. As a general rule, recommendation is to select 20 builds or 30 days at most.
* Select the check-box **Enable project-based security**. On the permission matrix below, ensure that anonymous users do not have any permission granted for the project and that project managers have all permissions. Add to the matrix as many users as needed:



The **Source Code Management** (SCM) section is where you can configure where Jenkins should look for the source code. When a build is triggered, Jenkins will delegate on a SCM plugin to download all the sources for a given location (or only those that have changed depending on configuration) and place them in a workspace folder where the actual build process will be launched.

SCM configuration depends greatly on the SCM system in use. By default Jenkins comes with CVS and Subversion support, but many others are available through the Jenkins update centre, as Rational ClearCase, Microsoft Team Foundation Server or Git. You can even use the File System SCM plugin to use a local or network folder as the origin of sources for a build. Let’s show a screenshot of an example of a job integrated with TFS:

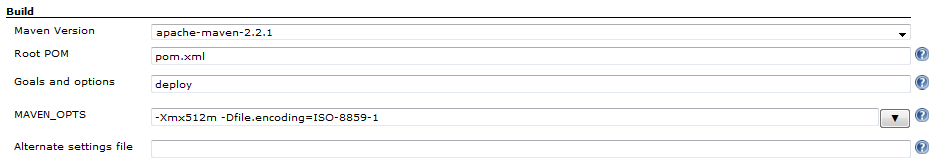


The **Build Triggers** section is where you can configure how or when the builds will be triggered. You can trigger builds when dependencies change, when source code is checked-in, periodically or manually (triggered from the dashboard or from external requests – even from Mobile phones!). To configure periodic builds a cron-like expression is used. For example, this example will trigger a build Mondays, Wednesdays and Fridays at 22:00 (10 PM):



For more information on the cron-like expressions used by Jenkins, click the help button to the right of the text input field.

The **Build** section, in Maven jobs, is where you can specify what Maven goals will be executed and advanced Maven options (for example more memory or external properties needed to run a build). When multiple Maven versions are configured, you may select which to use in this section. The screenshot below shows an example of a job running the deploy goal (included compilation, tests, packaging and deployment to an artefact repository) with some advanced Maven options:



The final two sections, **Build Settings** and **Post-build Actions**, depend greatly on the plugins installed in Jenkins. This is the place where you will interact typically with external tools as Sonar (for consolidated quality analysis – Continuous Quality Assurance), configure build notifications or enable release management tasks (for example deploy War artefact to application server and run assembly/regression tests).

Once all the configuration settings for the job has been entered, press the Save button at the bottom of the screen to finish:



# Build Results and Historic Information

Jenkins dashboard is a very capable tool to investigate and discover how your artefacts and builds are behaving with time. Out of the box you have almost everything you need to effectively manage your projects, and with the help of the available plugins Jenkins may adapt to all your needs.

You can configure dedicated views for projects or for users, focusing on what’s wrong, what is current status or trends. You can add historic/trend reports for test and quality metrics directly in the dashboard. When you integrate Jenkins with Sonar, you have access to the Sonar dashboard with one click.

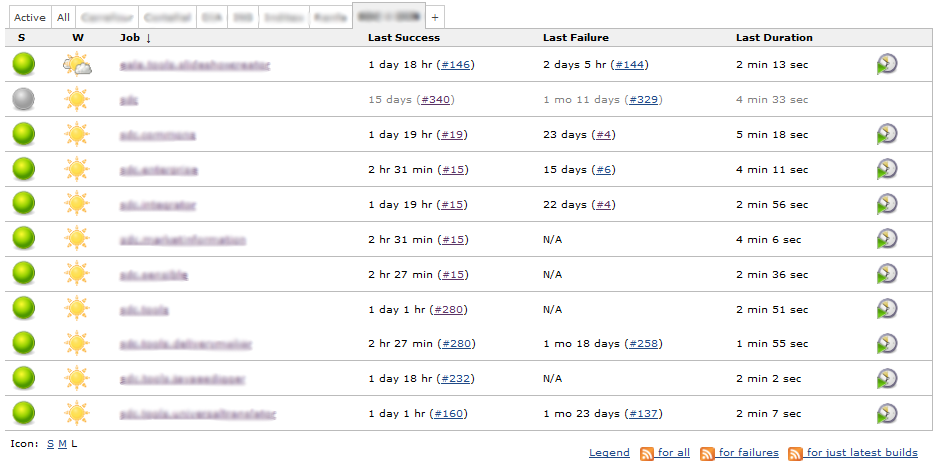
In this section we will describe the main features of the Jenkins dashboard to access and review build results and job historic information.

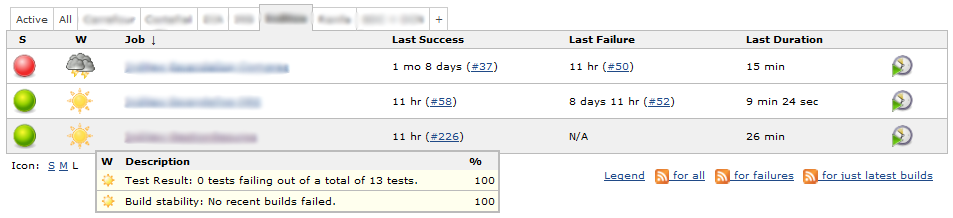
Once you setup a job in Jenkins, it will start launching builds as scheduled. For each build, Jenkins will follow the same procedure: get source code, run Maven goals (typically it will be Maven, although other build tools are available) and do post-build actions (send e-mail notifications, analyse code with Sonar, deploy to server for assembly/regression testing).

For each build, Jenkins registers its status from one of the three possible:

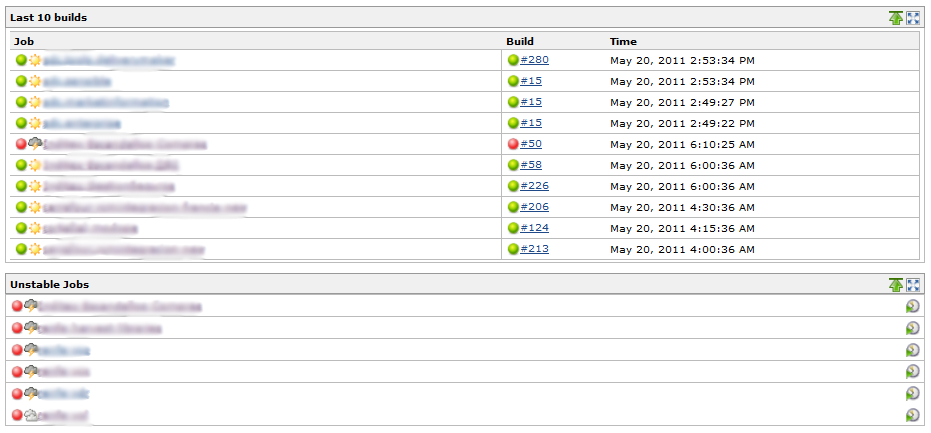
* Success/Stable: the build run and all tests finished successfully.
* Unstable: the build run, code compiles but some or all tests did not finish successfully.
* Failed: the build didn’t finish successfully, e.g. the code did not compile or some post-build action ended in error.

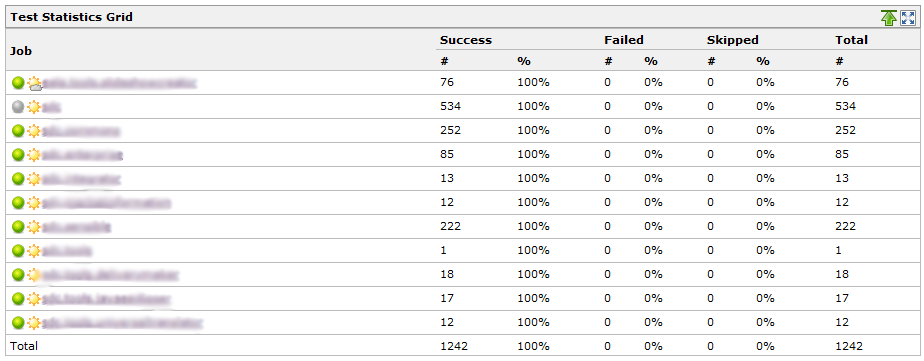
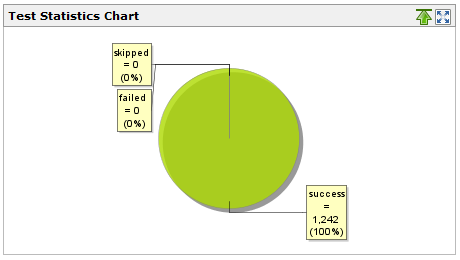
Jenkins dashboard shows the latest job status (blue/green, yellow, red balls) along with a ‘weather’ indicator showing the trend for latest five builds. At a glance, this information is very helpful to get a clear picture on what’s going on:



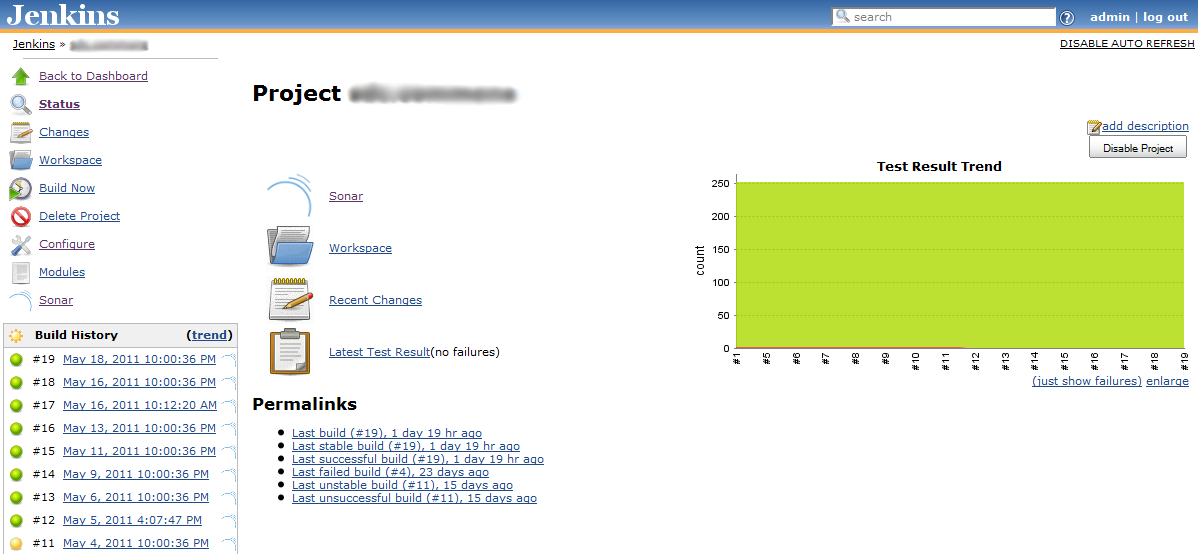


Other possible views that can be brought to the dashboard may show information on unstable/failed jobs only, or display blended test results for a group of jobs:



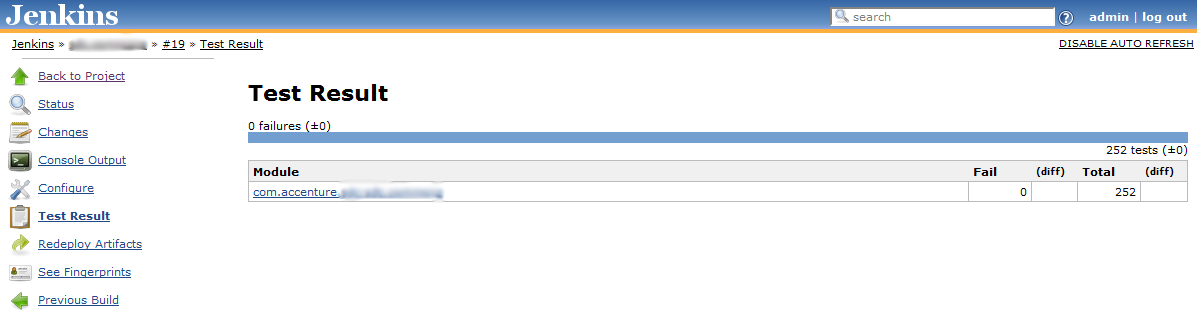
 

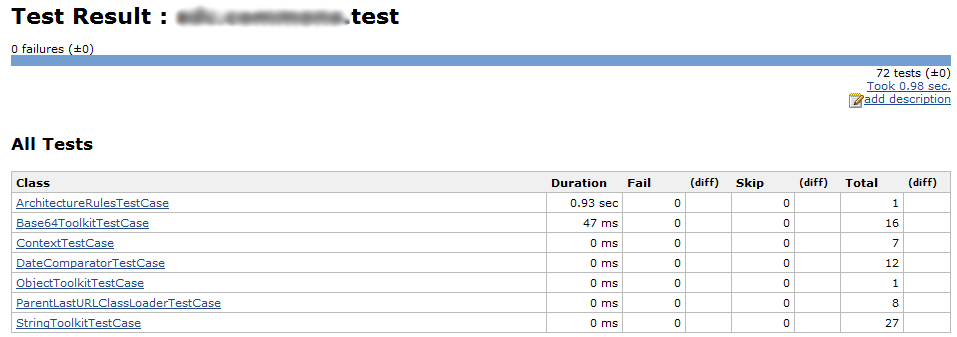
Given any view in the dashboard, when you click on the name of a job, you access the job status screen:

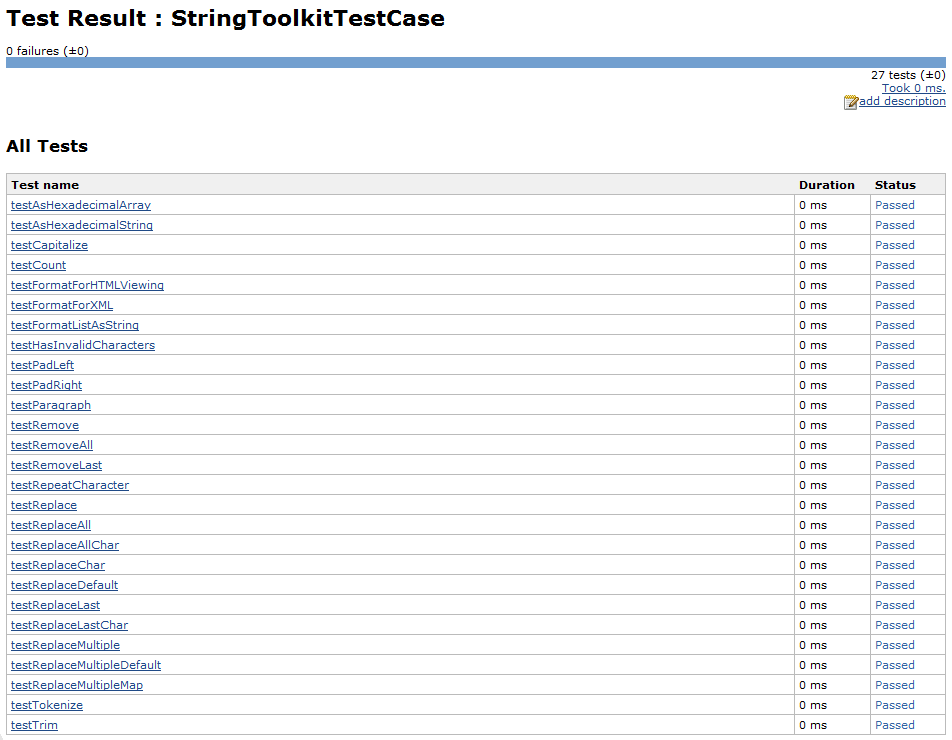


From this screen, you can access lots of useful information. We would highlight:

* **Changes**: Show you the changes done in SCM since previous build. This information is very useful to find out who has done what.
* **Workspace**: Lets you explore the files downloaded from SCM plus all working files generated during the build (e.g. Maven output). You can download the files individually or all together in a Zip with just one click.
* **Modules**: For Maven builds, information on the different modules that were executed during the build. It is useful to discover what module is causing a build failure, for example.
* **Sonar**: If you have integrated Jenkins and Sonar, this link will send you to the Sonar dashboard screen for this artefact. **Build History**: You can access to the information on any of the recent builds to explore what happened there, compare results, check the logs…
* **Test Results**: To the right you can see a diagram showing the test results and history. Clicking on the centre section link, in the right diagram or in the links below it will give you access to detailed information on the executed test, their status and deltas versus previous build. Details can be drilldown until single test level.







Once you click on a concrete build you can access more information, as the console output for this build, file fingerprints (more on fingerprints on section 6) or details on what Maven mojos (in the case of a Maven build) were executed:



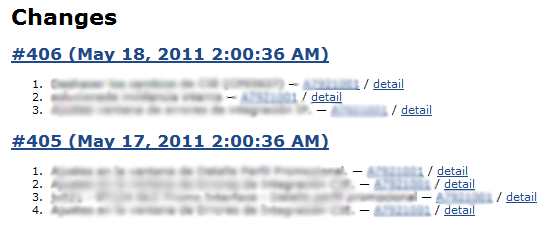
All this information, available at just a few clicks at the Jenkins dashboard, you will be able to manage and take control of your builds and artefacts, know what is happening anytime and plan actions when needed.

# Insights for Changes Between Builds

One of the most useful information shown in Jenkins dashboard is change information between builds.

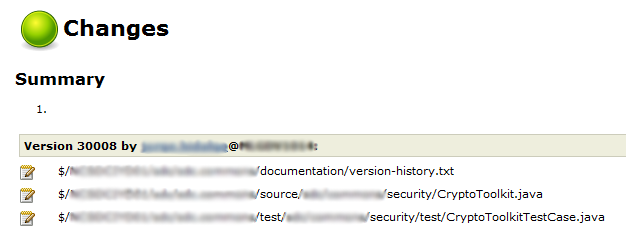
For each job build, Jenkins provides detailed insights on what happened at SCM level but also on dependencies level.

In the job status screen, as shown in previous section, you can access any recent build status and information, as well as information on changes between builds. When you click on **Changes** on the left menu or **Recent Changes** on the centre section, you access the summary of changes done in the SCM since previous build:

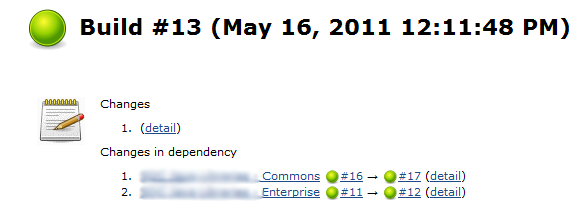
 

For each change set, you get the review number, timestamp, comments left by the developer in the SCM and the developer Id.

Clicking on the **detail** link will show details on the files modified in the change set:



Besides the changes in the SCM, Jenkins also shows information on updated dependencies since previous build. This information is shown once you enter the build detail screen:



Jenkins only tracks artefacts generated through configured jobs. That means that if your artefact has external dependencies you will not get this level of detail when they change.

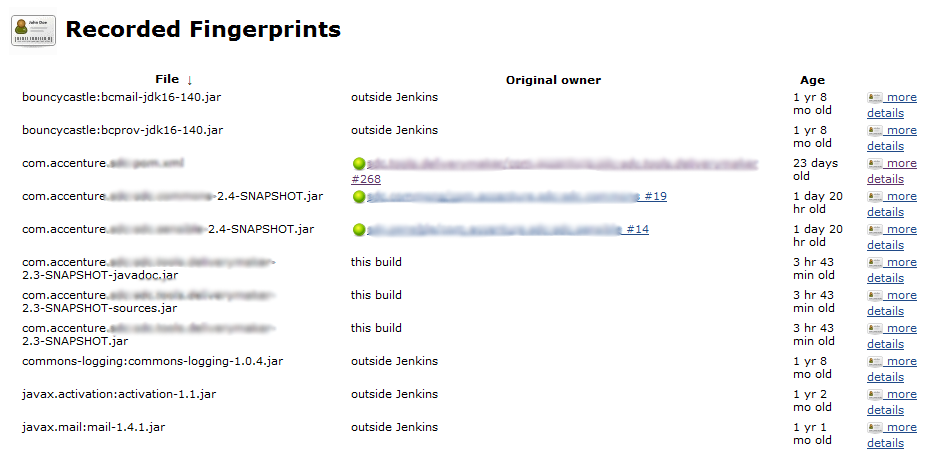
# Dependencies and Artefact Fingerprints

Another very useful feature of Jenkins is the tracking of dependencies and artefact usage across configured jobs.

When we discussed about changes in previous section, we showed that Jenkins is tracking what artefacts are used and where.

Every time that a Maven build is executed, Jenkins keeps a log of all the artefacts used in the build, and for those that were generated in another Jenkins job, it links the two of them. This feature is very useful to detect integration issues when an artefact is not compiling or behaving as expected due to a missing or wrong dependency with another artefact (or version).

When Jenkins logs the artefacts, it generates and stores a fingerprint, a hash that will identify the artefact from this moment onwards.



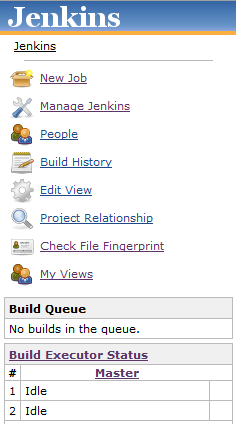
Moreover, you can click on any artefact and discover all other artefacts that are using it (or used it in the past):



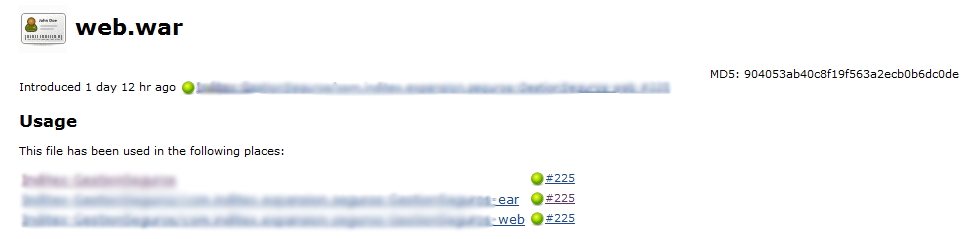
Now, let’s imagine the following scenario. You are running your application in the production environment and deploy some changes on it. After a few days, something starts to get wrong in the system and you need to track down the origin of this wrong behaviour. As some days have passed, you do not have a clear picture on what artefacts have changed; you only have a bunch of Jar/War files and would like to narrow down the search.

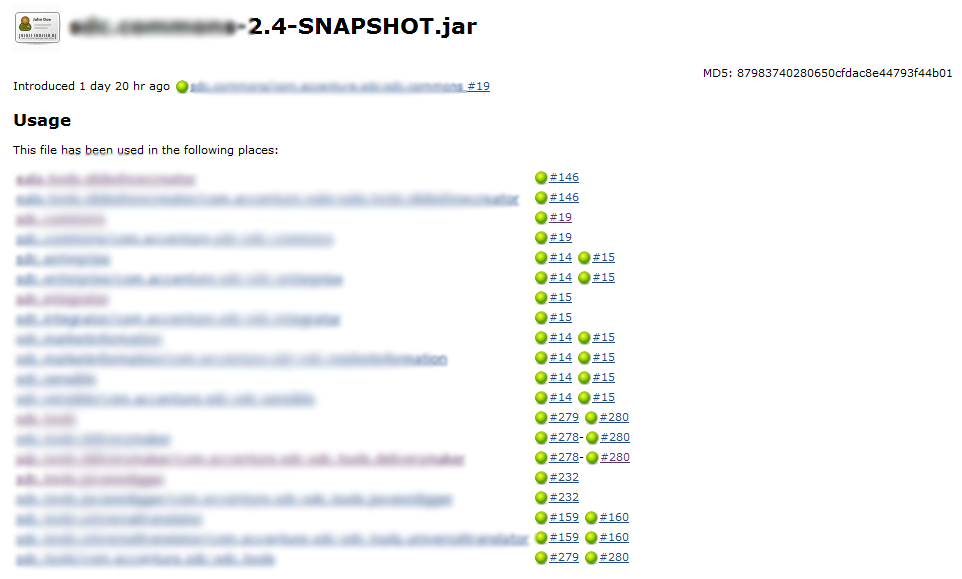
From Jenkins dashboard you can upload an artefact anytime and get information of it. You may be able to know whether the artefact has changed recently or not, and if it changed, when it was and what changes were introduced on this version. This information will prove useful to narrow down unexpected behaviours and speed up the fix of the issue.

To access this feature, just click **Check File Fingerprint** from the left menu on the home Jenkins screen. Then upload the artefact and press the button **Check**:

You will get the information on the artefact, if known, as when it was created and which artefacts are using it. You can access the version link to get insights on what changed on that concrete version and take action if needed:





# References

Jenkins: <http://jenkins-ci.org/>

Maven: <http://maven.apache.org/>

# Document Control

## Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Approver | Comment |
| 0.1 | 01/09/2015 | Sandip Mante |  | First draft version |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Open Issues

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Opening Version | Date | Author | Action | Owner | Status | Closing version |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |