

HOWTO Continuous Integration

Table of Contents

| | | |
|------|------------------------------------|----|
| 1. | Introduction to Jenkins CI | 1 |
| 1.1. | Jenkins main features | 2 |
| 1.2. | Jenkins installation..... | 2 |
| 2. | Using Jenkins..... | 6 |
| 2.1. | Administering Jenkins..... | 6 |
| 2.2. | Building a software project..... | 9 |
| 2.3. | Configuring automatic builds | 11 |
| 2.4. | Building a Maven2 project..... | 14 |
| 2.5. | Plugins in Jenkins | 16 |
| 3. | GitHub Actions..... | 16 |
| 4. | References | 19 |

1. Introduction to Jenkins CI

Jenkins is an application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. Among those things, current Jenkins focuses on the following two jobs:

1. **Building/testing software projects continuously**, just like CruiseControl or DamageControl. In a nutshell, Jenkins provides an easy-to-use so-called continuous integration system, making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. The automated, continuous build increases the productivity.
2. **Monitoring executions of externally-run jobs**, such as cron jobs and procmail jobs, even those that are run on a remote machine. For example, with cron, all you receive is regular e-mails that capture the output, and it is up to you to look at them diligently and notice when it broke. Jenkins keeps those outputs and makes it easy for you to notice when something is wrong.

HOWTO Continuous Integration

1.1. Jenkins main features

The main features of Jenkins are:

1. **Easy installation:** Just `java -jar jenkins.war --enable-future-java`, or deploy it in a servlet container. No additional install, no database.
2. **Easy configuration:** Jenkins can be configured entirely from its friendly web GUI with extensive on-the-fly error checks and inline help. There's no need to tweak XML manually anymore, although if you'd like to do so, you can do that, too.
3. **Change set support:** Jenkins can generate a list of changes made into the build from Subversion/CVS. This is also done in a fairly efficient fashion, to reduce the load on the repository.
4. **Permanent links:** Jenkins gives you clean readable URLs for most of its pages, including some permalinks like "latest build"/"latest successful build", so that they can be easily linked from elsewhere.
5. **RSS/E-mail/IM Integration:** Monitor build results by RSS or e-mail to get real-time notifications on failures.
6. **After-the-fact tagging:** Builds can be tagged long after builds are completed
7. **JUnit/TestNG test reporting:** JUnit test reports can be tabulated, summarized, and displayed with history information, such as when it started breaking, etc. History trend is plotted into a graph.
8. **Distributed builds:** Jenkins can distribute build/test loads to multiple computers. This lets you get the most out of those idle workstations sitting beneath developers' desks.
9. **File fingerprinting:** Jenkins can keep track of which build produced which jars, and which build is using which version of jars, and so on. This works even for jars that are produced outside Jenkins, and is ideal for projects to track dependency.
10. **Plugin Support:** Jenkins can be extended via 3rd party plugins. You can write plugins to make Jenkins support tools/processes that your team uses.

1.2. Jenkins installation

As already mentioned installing Jenkins is extremely simple. You have to follow these two steps:

1. Download `jenkins.war` from <http://mirrors.jenkins-ci.org/war/latest/jenkins.war>
2. Execute in command line the statement: `java -jar jenkins.war`. Notice that some of you may have port 8080 taken by another server. Therefore, you may launch it with the following configuration: `java -jar jenkins.war --httpPort=7777`

During the installation process, an admin user is created and a password generated. You must use that password in the installation process. You will have the chance to `Install suggested plugins` and to create an admin user (and change the password), as part of the initialization process.

HOWTO Continuous Integration

```
C:\Windows\System32\cmd.exe - java -jar jenkins.war

of factory hierarchy
abr 24, 2018 12:01:52 PM hudson.model.DownloadService$Downloadable load
INFORMACIEN: Obtained the updated data file for hudson.tasks.Maven.MavenInstall
r
abr 24, 2018 12:01:55 PM hudson.model.DownloadService$Downloadable load
INFORMACIEN: Obtained the updated data file for hudson.tools.JDKInstaller
abr 24, 2018 12:01:56 PM hudson.model.AsyncPeriodicWork$1 run
INFORMACIEN: Finished Download metadata. 26.041 ms
abr 24, 2018 12:02:04 PM jenkins.install.SetupWizard init
INFORMACIEN:

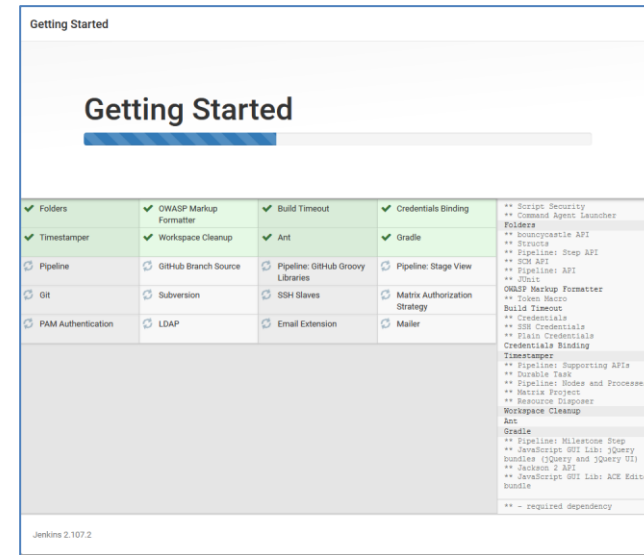
*****
*****
*****

Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:

c3d8adb3e56b439499b70e15f6eb48f9

This may also be found at: C:\Users\rebeca.cortazar\.jenkins\secrets\initialAdminPassword

*****
*****
```



In order to access Jenkins web interface, you must open a browser and go to <http://localhost:8080>.

You might want to run Jenkins as a service, so it starts up automatically without requiring a user to log in. For that within the Jenkins → Manage Jenkins menu click on the option 'Install as Windows Service'. For more details go to: <https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service>. Alternatively, you can install a servlet container like GlassFish and Tomcat, which can run as a service by itself, and then deploy Jenkins to it.

Notice that Git comes configured as SCM by default. However, in order to add more plugins to Jenkins, follow these steps:

- Select the menu option `Jenkins`→`Manage Jenkins` which appears on the top left hand side when you access to Jenkins web interface
- Select the option `Manage Plugins` as shown in Figure 1. More details on plugin installation in section 2.5.
- Click on the `Available plugins` tab as shown in Figure 2.
- Check the `Maven Integration Plugin` option and choose “`Install without restart`” in the `Available Plugins Window` (see Figure 3)

HOWTO Continuous Integration

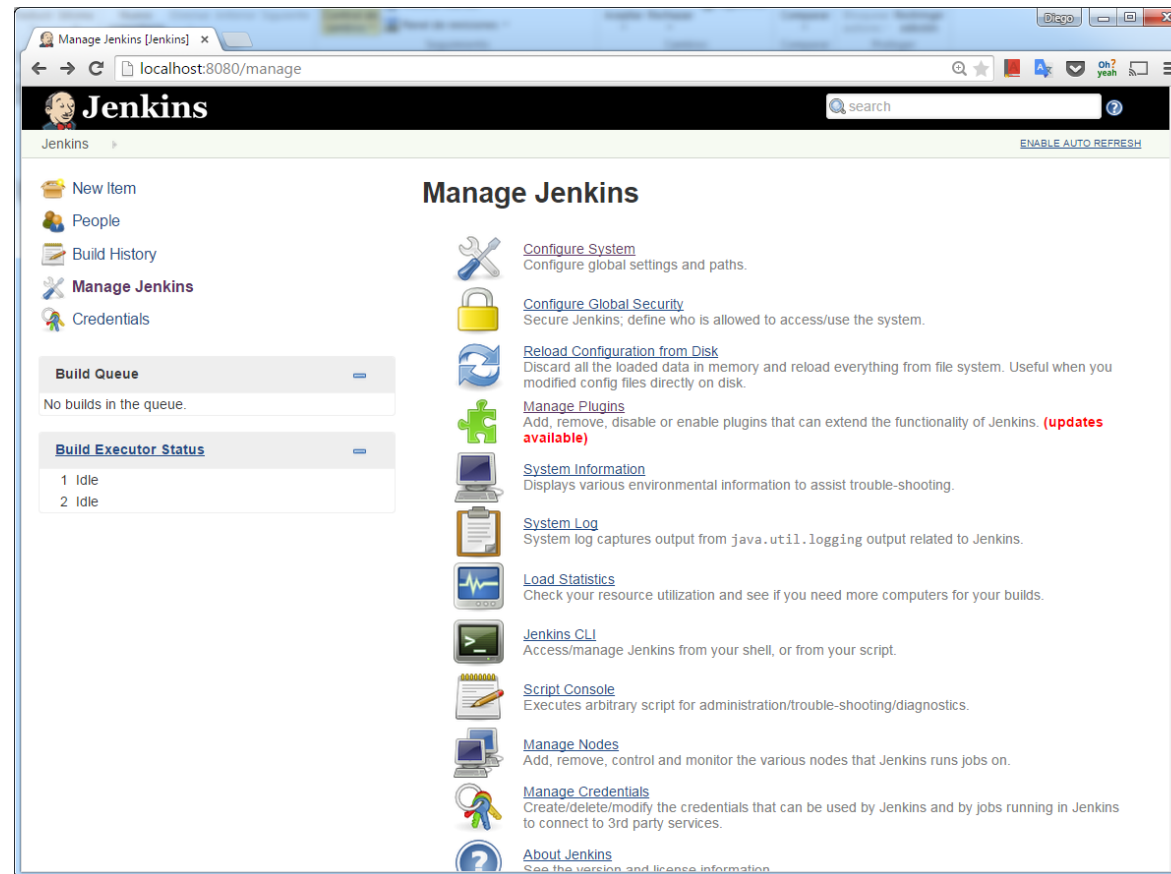


Figure 1. Manage Jenkins screen in Jenkins

HOWTO Continuous Integration

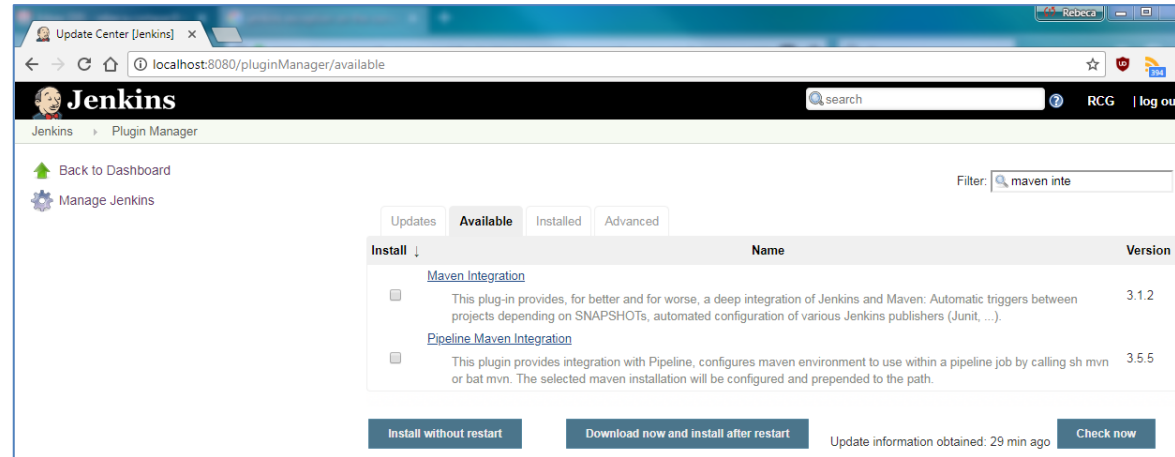


Figure 2. Manage Plugins, Available tab window within Manage Jenkins menu

| Updates | Available | Installed | Advanced |
|---------------------------------|--|-----------|----------|
| Install | Name | Version | |
| .NET Development | | | |
| <input type="checkbox"/> | CCM This plug-in generates the trend report for CCM, an open source static code analysis program. | 3.2 | |
| <input type="checkbox"/> | ExCop Runner | 1.1 | |
| <input type="checkbox"/> | MSBuild Allows using MSBuild to build .NET projects | 1.29 | |
| <input type="checkbox"/> | MSTest Generates test reports for MSTest. | 0.23 | |
| <input type="checkbox"/> | MSTestRunner | 1.3.0 | |
| <input type="checkbox"/> | NAnt | 1.4.3 | |
| <input type="checkbox"/> | PowerShell | 1.3 | |
| <input type="checkbox"/> | Violation Comments to Bitbucket Server Finds violations reported by code analyzers and comments Bitbucket Server (or Stash) pull requests (or commits) with them. | 1.70 | |
| <input type="checkbox"/> | Violations | 0.7.11 | |
| <input type="checkbox"/> | Visual Studio Code Metrics | 1.7 | |
| <input type="checkbox"/> | VSTest Runner | 1.0.7 | |
| <input type="checkbox"/> | Start Windocks Containers Allows for the creation and start of Windocks Containers | 1.4 | |
| <input type="checkbox"/> | change-assembly-version-plugin | 1.10 | |
| Agent Launchers and Controllers | | | |
| <input type="checkbox"/> | Amazon EC2 Container Service Jenkins plugin to run dynamic slaves in a Amazon ECS/Docker environment | 1.14 | |
| <input type="checkbox"/> | Compound Slaves | 1.2 | |

Figure 3. Available plugins windows

HOWTO Continuous Integration

2. Using Jenkins

Once Jenkins has been installed you access to it by going to the URL: <http://myServer:8080> where `myServer` is the name of the system running Jenkins. Remember that you can specify a different port by using `--httpPort=$HTTP_PORT` where `$HTTP_PORT` is the port you want Jenkins to run on.

2.1. Administering Jenkins

Jenkins needs some disk space to perform builds and keep archives. You can check this location from the configuration screen of Jenkins. By default, this is set to `~/.jenkins`, this is the folder `c:\users\<username>\.jenkins`, but you can change this in one of the following ways:

- Set "JENKINS_HOME" environment variable to the new home directory before launching the servlet container.
- Set "JENKINS_HOME" system property to the servlet container.
- Set JNDI environment entry "JENKINS_HOME" to the new directory.

The directory structure of JENKINS_HOME is as follows. The text under braces explains the purpose of each directory/file.

```
JENKINS_HOME
+- config.xml      (jenkins root configuration)
+- *.xml           (other site-wide configuration files)
+- userContent     (files in this directory will be served under your http://server/userContent/)
+- fingerprints    (stores fingerprint records)
+- plugins         (stores plugins)
+- jobs
  +- [JOBNAME]     (sub directory for each job)
    +- config.xml  (job configuration file)
    +- workspace   (working directory for the version control system)
    +- latest      (symbolic link to the last successful build)
    +- builds
      +- [BUILD_ID] (for each build)
        +- build.xml (build result summary)
        +- log       (log file)
        +- changelog.xml (change log)
```

HOWTO Continuous Integration

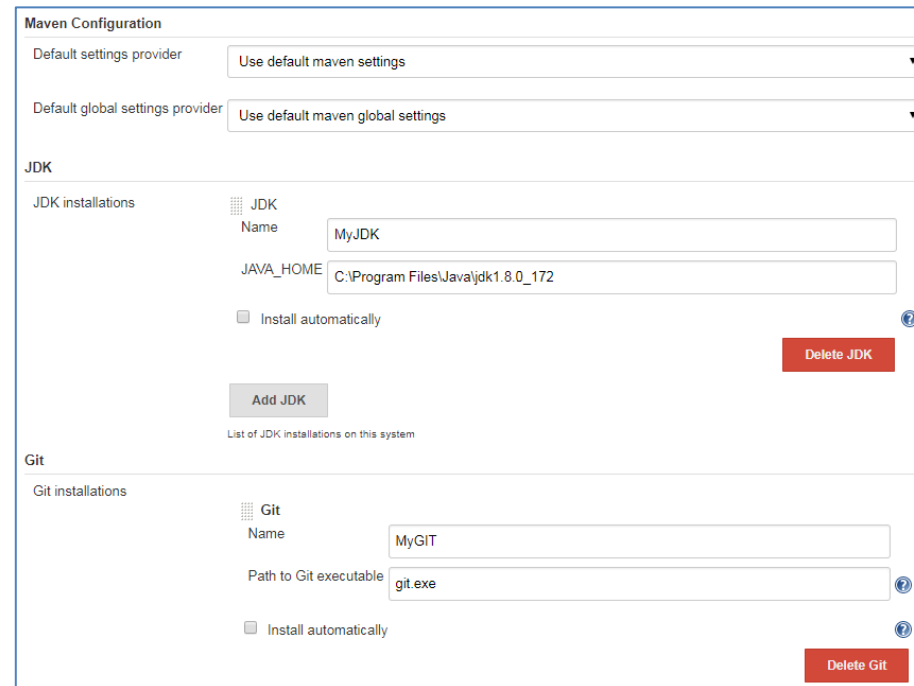
You can change this location after you've used Jenkins for a while, too. To do this, stop Jenkins completely, move the contents from old `JENKINS_HOME` to the new home, set the new `JENKINS_HOME`, and restart Jenkins.

For more details on backing up Jenkins, moving/copying/renaming jobs please visit <https://wiki.jenkins-ci.org/display/JENKINS/Administering+Jenkins>.

Before you can start configuring and launching jobs managed by Jenkins you need to configure your Jenkins installation. For that, you need to select Jenkins → Manage Jenkins. Once there:

- Got to `Global Tool Configuration` to configure JDK, Git and Maven by indicating where you have these tools installed in your machine (see Figure 4 and Figure 5)
- Go to `Configure System` menu option and configure the SMTP server details for email notification for build success or failure reporting (see Figure 6). If you have 2-step verification activated in your google account, you must generate an Apps password for Jenkins. Details: <https://support.google.com/accounts/answer/185833?hl=en>

Jenkins has a set of built-in plugins that allow code building with Ant or Maven, SCM access through SVN or Git or report notification through email.

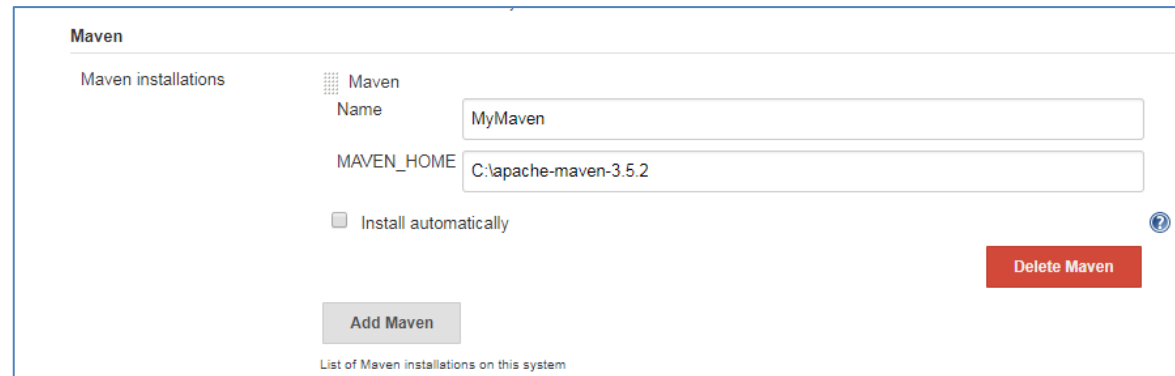


The screenshot displays the 'Global Tool Configuration' window in Jenkins, which is organized into three main sections: Maven Configuration, JDK, and Git.

- Maven Configuration:** This section at the top contains two dropdown menus. The 'Default settings provider' is set to 'Use default maven settings', and the 'Default global settings provider' is set to 'Use default maven global settings'.
- JDK:** This section is for configuring Java Development Kits. It features a table with one entry named 'MyJDK'. The 'JAVA_HOME' for this entry is 'C:\Program Files\Java\jdk1.8.0_172'. There is an unchecked checkbox for 'Install automatically'. To the right of the entry is a red 'Delete JDK' button. Below the table is a grey 'Add JDK' button and a small text label 'List of JDK installations on this system'.
- Git:** This section is for configuring Git. It features a table with one entry named 'MyGIT'. The 'Path to Git executable' is 'git.exe'. There is an unchecked checkbox for 'Install automatically'. To the right of the entry is a red 'Delete Git' button.

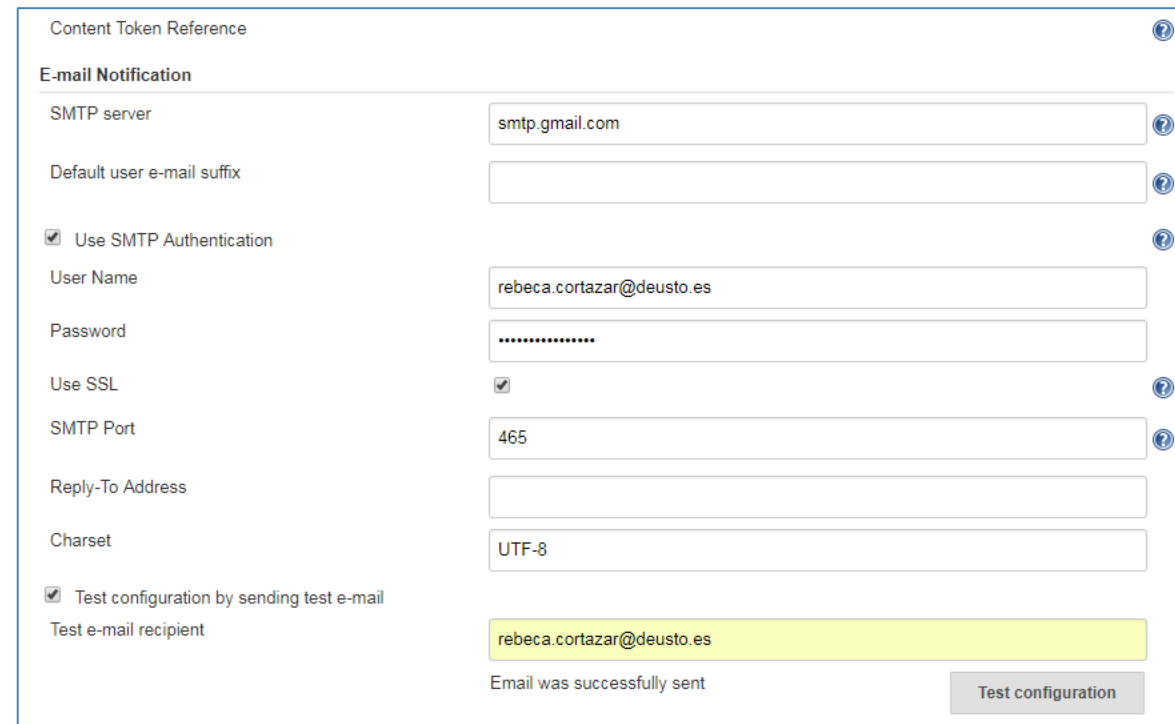
Figure 4. Global Tool Configuration window within Manage Jenkins (JDK and Git Configuration)

HOWTO Continuous Integration



The image shows the 'Maven' configuration window in Jenkins. It has a title bar 'Maven' and a sub-header 'Maven installations'. Below this, there's a 'Maven Name' field with the value 'MyMaven' and a 'MAVEN_HOME' field with the value 'C:\apache-maven-3.5.2'. There's an unchecked checkbox for 'Install automatically'. At the bottom left is an 'Add Maven' button, and at the bottom right is a red 'Delete Maven' button. A small blue question mark icon is next to the 'Delete Maven' button. At the very bottom, there's a link 'List of Maven installations on this system'.

Figure 5. Global Tool Configuration window within Manage Jenkins (Maven configuration)



The image shows the 'E-mail Notification' configuration window in Jenkins. It has a title bar 'Content Token Reference' and a sub-header 'E-mail Notification'. Below this, there's an 'SMTP server' field with the value 'smtp.gmail.com'. There's a 'Default user e-mail suffix' field. There's a checked checkbox for 'Use SMTP Authentication'. Below this, there's a 'User Name' field with the value 'rebeca.cortazar@deusto.es' and a 'Password' field with masked characters. There's a checked checkbox for 'Use SSL'. Below this, there's an 'SMTP Port' field with the value '465'. There's a 'Reply-To Address' field. There's a 'Charset' field with the value 'UTF-8'. There's a checked checkbox for 'Test configuration by sending test e-mail'. Below this, there's a 'Test e-mail recipient' field with the value 'rebeca.cortazar@deusto.es'. At the bottom, there's a message 'Email was successfully sent' and a 'Test configuration' button.

Figure 6. Configure System window within Manage Jenkins (email configuration)

HOWTO Continuous Integration

2.2. Building a software project

Jenkins can be used to perform the typical build server work, such as doing continuous/official/nightly builds, run tests, or perform some repetitive batch tasks. This is called "free-style software project" in Jenkins.

To test it (see Figure 7), go to Jenkins top page, select "New Job", then choose "Build a free-style software project". This job type consists of the following elements:

- Optional SCM, such as CVS, Subversion or GIT where your source code resides.
- Optional triggers to control when Jenkins will perform builds.
- Some sort of build script that performs the build (ant, maven, shell script, batch file, etc.) where the real work happens
- Optional steps to collect information out of the build, such as archiving the artifacts and/or recording javadoc and test results.
- Optional steps to notify other people/systems with the build result, such as sending e-mails, IMs, updating issue tracker, etc.

Let's exercise this with the project `MoneyMoneyPerf`, used in the last Performance Testing HOWTO. As a prerequisite, you should upload into a GIT repository the contents of that lab session's code. For instance, upload the projects files into at your Git repository, e.g.

[https://github.com/\[yourGitHubaccount\]/MoneyMoneyJenkins.git](https://github.com/[yourGitHubaccount]/MoneyMoneyJenkins.git). And then clone them in another folder. The steps to apply would be the following:

- `mkdir jenkins_test`
- `cd jenkins_test`
- `git clone https://github.com/\[yourGitHubaccount\]/MoneyMoneyJenkins.git`

The steps to be followed to configure a new job in Jenkins which makes use of Maven and Git are:

1. Assign a name to the free-style software project. For instance, `MoneyMoneyJenkins_Test`.
2. Select Git as the option for Source Code Management (SCM). Type in the URL to the repository. See Figure 8 for more details. You can configure the branch you want to pull from the remote repository (by default, master). Observe in Figure 9 that we have also added a `Credential` in order to access a Git repository.
3. Indicate a trigger when the project should be built. For instance, every time changes occur in the SCM. For that, select `Poll SCM` and type in the Schedule box a valid cron expression, e.g. `H/15 * * * *` (every fifteen minutes). That way you may configure automatic builds.
4. Configure Maven build goal and indicate the location of the `pom.xml` within the pulled project from the Git repository, as show in Figure 10.
5. State a valid post-build option. For instance, enable E-mail notification of the integration build process. Choose among the available options and fill in the emails of people that should receive the reports (see Figure 10).
6. Click on Save at the bottom of the job configuration screen.

HOWTO Continuous Integration

7. Go to the Jenkins main page and click on the build now icon, on the right hand side of each job listing. See Figure 12. An icon will start blinking passing from the in-progress state (blinking blue) to success state (blue) or failure end state (red). Notice that you might configure the project to be built regularly by configuring an automatic built as soon in Figure 11.

Figure 7. New Job configuration screen.

When a Jenkins job executes, it sets some environment variables that you may use in your shell script, batch command, Ant script or Maven POM. Maven requires that you include the parameter as part of the build goals. An example Jenkins configuration for the Maven "Goals" field would be: `clean install -DBUILD_NUMBER=${BUILD_NUMBER}`

The following table contains a list of all of these environment variables.

| Environment Variable | Description |
|----------------------|--|
| BUILD_NUMBER | The current build number, such as "153" |
| BUILD_ID | The current build id, such as "2005-08-22_23-59-59" (YYYY-MM-DD_hh-mm-ss) |
| BUILD_URL | The URL where the results of this build can be found (e.g. http://buildserver/jenkins/job/MyJobName/666/) |
| JOB_NAME | Name of the project of this build. This is the name you gave your job when you first set it up. It's the third column of the Jenkins Dashboard main page. |
| BUILD_TAG | String of jenkins-\${JOBNAME}-\${BUILD_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification. |
| JENKINS_URL | Set to the URL of the Jenkins master that's running the build. This value is used by Jenkins CLI for example |

HOWTO Continuous Integration

| | |
|-----------------|--|
| EXECUTOR_NUMBER | The unique number that identifies the current executor (among executors of the same machine) that's carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1. |
| JAVA_HOME | If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to have \$JAVA_HOME/bin. |
| WORKSPACE | The absolute path of the workspace. |
| SVN_REVISION | For Subversion-based projects, this variable contains the revision number of the module. If you have more than one module specified, this won't be set. |
| CVS_BRANCH | For CVS-based projects, this variable contains the branch of the module. If CVS is configured to check out the trunk, this environment variable will not be set. |
| GIT_COMMIT | For Git-based projects, this variable contains the Gitish of the commit checked out for the build |
| GIT_BRANCH | For Git-based projects, this variable contains the Git branch that was checked out for the build (normally master) |

2.3. Configuring automatic builds

Builds in Jenkins can be triggered periodically (on a schedule, specified in configuration), or when source changes in the project have been detected, or they can be automatically triggered by requesting a URL:

- You could make Jenkins poll your Revision Control System for changes. See 'Poll SCM' option in bottom part of Figure 11.
- You can specify how often Jenkins polls your revision control system using the same syntax as `crontab` on Unix/Linux. Thus, your project may be executed periodically.
 - `cron` is a time-based job scheduler in Unix-like computer operating systems. It enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email.
 - Figure 11 shows an example of how to configure Jenkins to run the build every 6 minutes through the expression:
 - `6,12,18,24,30,36,42,48,54,0 * * * *`
 - For more information about `cron` check: <http://en.wikipedia.org/wiki/Cron>
- You can simply access to a URL with the following format to launch a job execution: <http://YOURHOST/jenkins/job/PROJECTNAME/build>. E.g. <http://localhost:8080/job/MoneyMoneyJenkins/build>. In this case, it executes the `MoneyMoneyJenkins` job.
- Alternatively, you may request through a URL that Jenkins checks the SCM to figure out whether a build should be launched (in case of encountering changes) or not: <http://YOURHOST/jenkins/job/PROJECTNAME/polling>. E.g. <http://localhost:8080/job/MoneyMoneyJenkins/polling>.

HOWTO Continuous Integration

- You can even make Jenkins to launch jobs by sending an email to it.

Notice that periodic building should not be used extensively. When people first start continuous integration, they are often so used to the idea of regularly scheduled builds like nightly/weekly that they use this feature. However, the point of continuous integration is to start a build as soon as a change is made, to provide a quick feedback to the change. To do that you should hook up SCM change notification to Jenkins. That is what the 'Poll SCM' option in Figure 11 enables. It allows you to periodically check for code changes, and only when those changes exist the build process is launched.

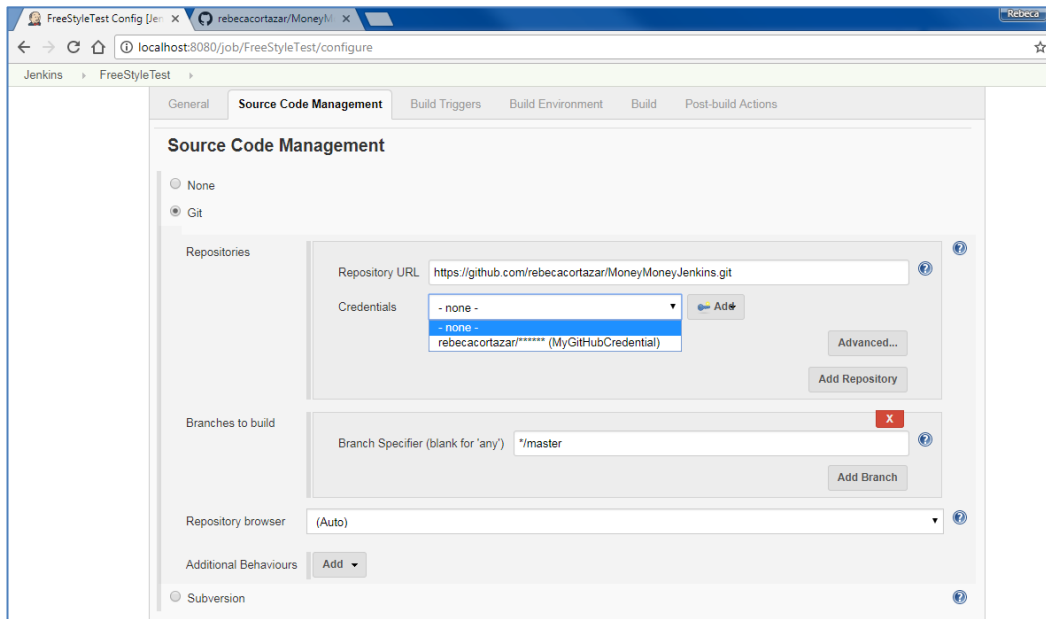


Figure 8. MoneyMoneyJenkins project configuration (top part of the screen)

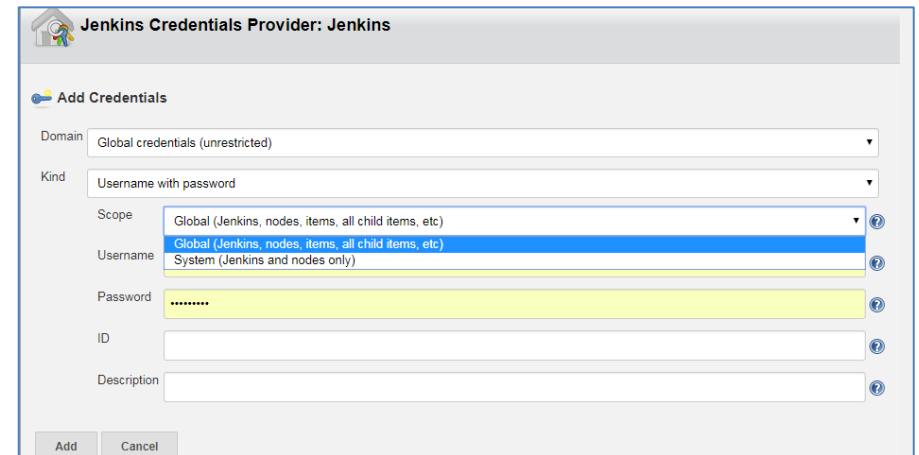


Figure 9. MoneyMoneyJenkins project configuration (adding Credentials – username and password kind)

HOWTO Continuous Integration

The screenshot displays the Jenkins configuration interface for a project. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment, **Build**, and Post-build Actions. The **Build** tab is selected, showing the 'Build' section with the following configuration:

- Invoke top-level Maven targets** (with a red 'X' icon and a help icon):
 - Maven Version**: MyMaven
 - Goals**: test
 - POM**: (empty text area)
 - Properties**: (empty text area)
 - JVM Options**: (empty text area)
 - Inject build variables**: ☐
 - Use private Maven repository**: ☐
 - Settings file**: Use default maven settings
 - Global Settings file**: Use default maven global settings

Below the 'Build' section is the 'Post-build Actions' section, which includes an 'E-mail Notification' action (with a red 'X' icon and a help icon):

- Recipients**: rebeca.cortazar@deusto.es
- Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.
- ☒ Send e-mail for every unstable build
- ☐ Send separate e-mails to individuals who broke the build

At the bottom of the configuration page, there are 'Save' and 'Apply' buttons.

Figure 10. MoneyMoneyJenkins project configuration (Maven configuration and bottom part of the screen)

HOWTO Continuous Integration

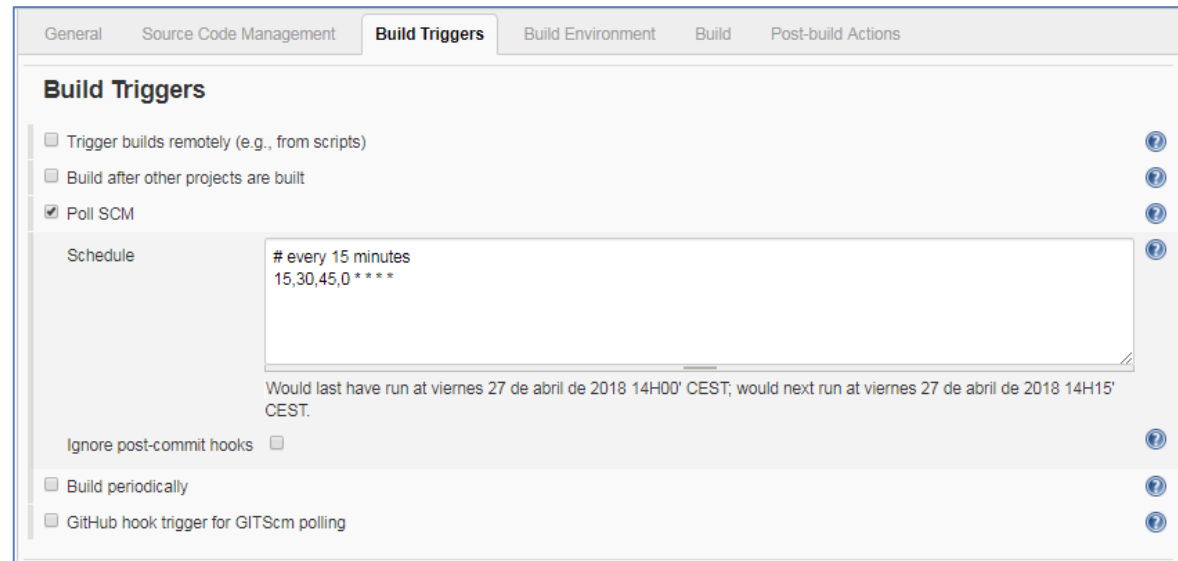


Figure 11. Automatic Build configuration in Jenkins though Menu option <jobname>/Configure

2.4. Building a Maven2 project

Jenkins provides a job type dedicated to Maven 2/3 (see Figure 13). This job type integrates Jenkins deeply with Maven 2/3 and provides the following benefits compared to the more generic free-style software project.

- Jenkins parses Maven POMs to obtain much of the information needed to do its work. As a result, the amount of configuration is drastically reduced.
- Jenkins listens to Maven execution and figures out what should be done when on its own. For example, it will automatically record the JUnit report when Maven runs the test phase. Or if you run the javadoc goal, Jenkins will automatically record javadoc.
- Jenkins automatically creates project dependencies between projects which declare SNAPSHOT dependencies between each other. See below.

Thus, mostly you just need to configure SCM information and what goals you'd like to run, and Jenkins will figure out everything else.

HOWTO Continuous Integration

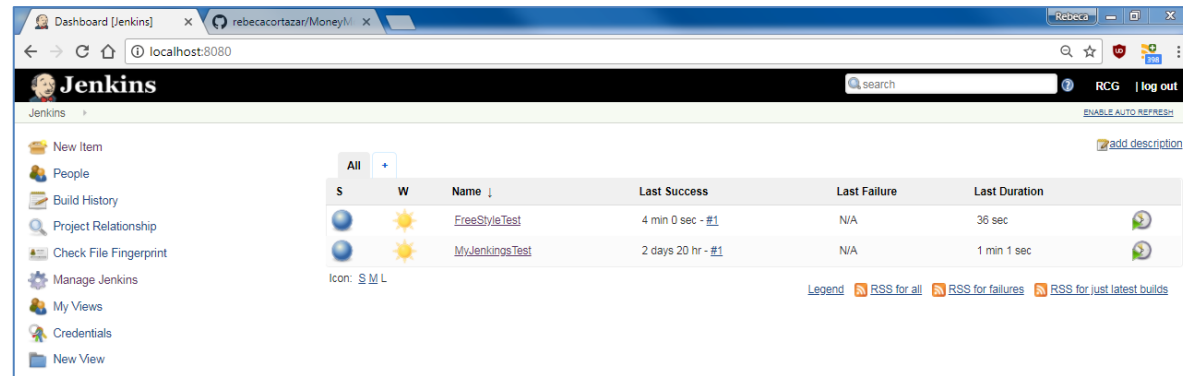





Figure 12. Jenkins front-page

Enter an item name

» This field cannot be empty, please enter a valid name

 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

 **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


 **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Figure 13. Jenkins configuration of a Maven job

HOWTO Continuous Integration

2.5. Plugins in Jenkins

Jenkins defines extensibility points, which are interfaces or abstract classes that model an aspect of a build system. You can add, remove, disable or enable plugins that can extend the functionality of Jenkins. For instance, you could install the Twitter notification plugin to enable Post-build notification through Twitter. Use the menu option Jenkins→Manage Jenkins→Manage Plugins to enable/disable plugins.

3. GitHub Actions

GitHub Actions automate, customize, and execute your software development workflows right in your GitHub repository. With them, you can discover, create, and share actions to perform any job you would like, including CI/CD, and combine actions in a completely customized workflow. Full documentation is available at: <https://docs.github.com/en/actions>

The steps to follow to learn how to use GitHub Actions are:

- 1) Read the GitHub Actions Quickstart: <https://docs.github.com/en/actions/quickstart>
- 2) Read the tutorial on how to launch Maven powered workflows through GitHub Actions: <https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven>
- 3) Check the syntax to define new configuration files for GitHub actions at: <https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions>
- 4) Use the example already facilitated in ALUD under the example entitled JenkinsCoberturaGitHubActions, file `.github/workflows/maven.yml`. To launch it, simply deploy the contents of folder JenkinsCoberturaGitHubActions into a GitHub repo.

HOWTO Continuous Integration

Actions · dipina/JenkinsCobertura

github.com/dipina/JenkinsCobertura/actions/workflows/maven.yml

0.0.0.0 Read It Later googlemaps - Goo... Image result for ed... 103b, 105, 106 - 22... See original image Semantic DataBase Get to Know Us | T... Educación - Depart... New Tab Deusto - Google M... Para todos los usua... artime eng » Other bookmarks

Search or jump to...

Pull requestsIssuesMarketplaceExplore

dipina/JenkinsCoberturaPublic

PinUnwatch1Fork0Star0

<> CodeIssuesPull requests1ActionsProjectsWikiSecurityInsightsSettings

WorkflowsNew workflow

All workflowsJava CI with Maven

Java CI with Mavenmaven.yml

Filter workflow runs

5 workflow runs

EventStatusBranchActor

enhanced README file

Java CI with Maven #5: Commit 33234cf pushed by dipina

master

41 minutes ago19s

corrected mistake of thread sleepo

Java CI with Maven #4: Commit b44105f pushed by dipina

master

1 hour ago19s

adding a too long delay to make tests to fail

Java CI with Maven #3: Commit 34a1e2e pushed by dipina

master

1 hour ago31s

Updated maven.yml

Java CI with Maven #2: Commit 8750bbb pushed by dipina

master

1 hour ago20s

Create maven.yml

Java CI with Maven #1: Commit eecc508 pushed by dipina

master

1 hour ago32s

Figure 14. GitHub action configured in a GitHub repo.

HOWTO Continuous Integration

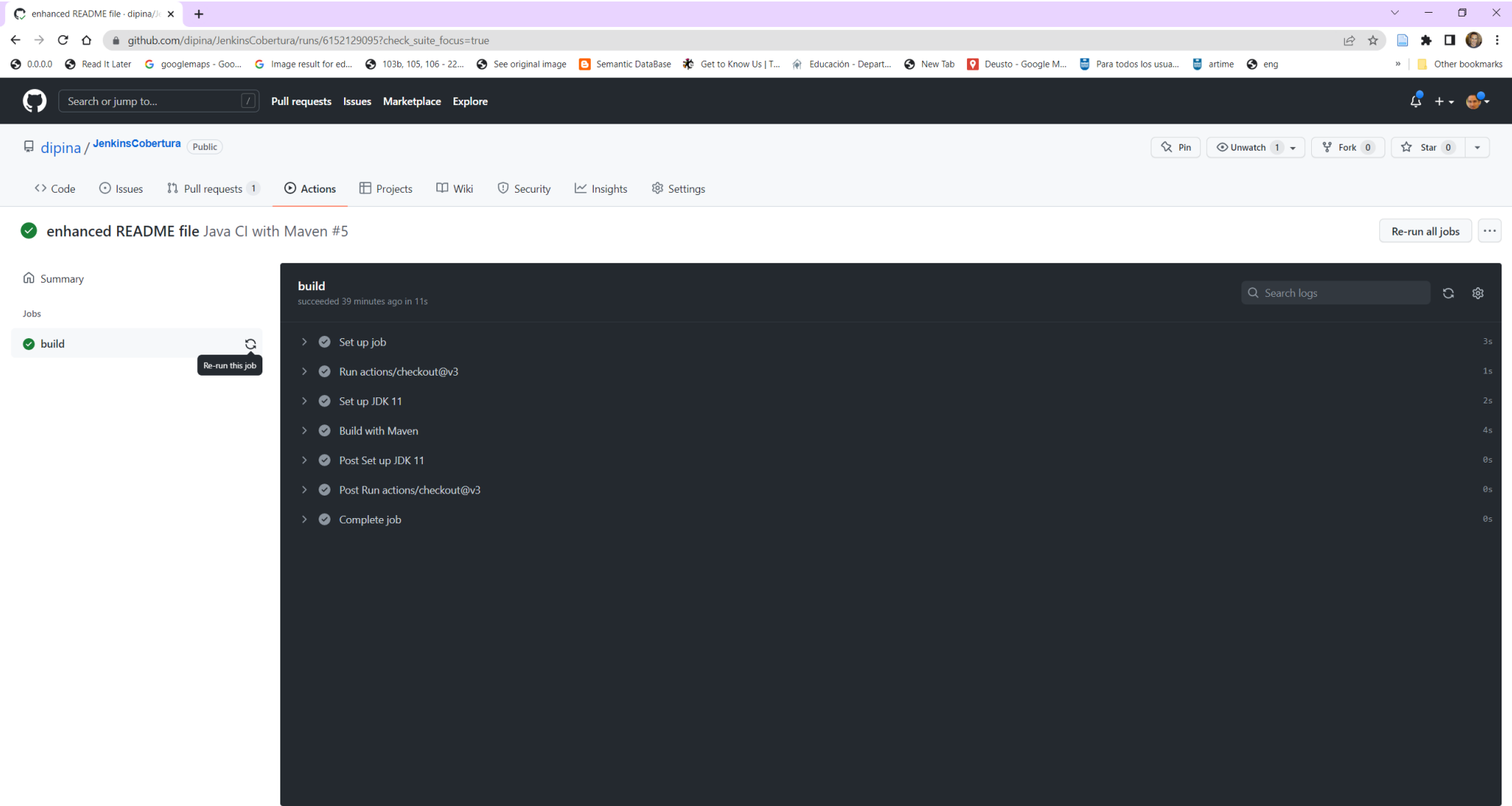


Figure 15. GitHub action execution example.

HOWTO Continuous Integration

4. References

- Meet Jenkins, <https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>
- Use Jenkins, <https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins>
- Meet GitHub Actions, <https://docs.github.com/en/actions>