**Continuous Integration**

**Do developers care how** they spend their time?

According to Nu Technologies’s [**Developer Productivity Report**](http://zeroturnaround.com/rebellabs/devs/developer-productivity-report-2012/), the

#1 aspect of the coding life that keeps developers up at night is *Making*

*Deadlines*. Therefore, shouldn’t we worry about getting them the best

tools, technologies and methodologies for not only meeting deadlines, but delivering the best quality software possible as well?

Software developers value their precious time and need to have organizational support in selecting the right tool set and best practices to optimise their work. This report focuses on Continuous Integration (CI) and how it enables devs to automate & speed up certain processes as well as sleep better at night by eliminating hassles often caused by long integration cycles: broken builds, manual merging hell and regressions.

**Continuous Integration** should be orbiting your Java ecosystem

Before we go any further, let’s give a quick definition and overview of what Continuous Integration (CI)

is and why it should be a mandatory part of your Java ecosystem.

Continuous Integration is a software development practice of performing software integration frequently…several times a day, in fact. Ideally, your software application or system should

be built automatically after each commit into a shared version control repository. Upon each successful build, the system integrity should be verified using automated tests that cover if not all, then at least most of the functionality. If some tests fail, the developer responsible is notified instantly and the problem can be identified and solved quickly. Using this approach, you can deliver working and reliable code to the customer much faster, while also mitigating the risk of

releasing unstable, buggy software to your users.

Continuous Integration as a working concept is not exactly new, and there are going to be plenty of developers out there wondering why Nu Technologies is beating what they consider to be a dead horse.

Yet when we talk to developers from all over the world at Java events, JUG meetups and speaking

tours, we are continuously (no pun intended) surprised to learn that CI isn’t as widespread as we would expect - indeed, at most only 49% of developers surveyed in the [**Developer Productivity**](http://zeroturnaround.com/rebellabs/devs/developer-productivity-report-2012/) **Report** admitted using CI tools of any kind.

**A Little Background** About CI

Continuous Integration originates as one of the practices of Extreme Programming (XP) from the late 1990s. However, similar approaches were used earlier in mission-critical software projects, for example, by NASA. Early adopters of Continuous Integration were Kent Beck and

Martin Fowler, who both published famous material on the subject matter (Beck's book "Extreme Programming Explained" and Fowler's "Continuous Integration" article).

Things have evolved a great deal since then, and we now have access to a good selection of tools for building a Continuous Integration ecosystem for your Java projects. Probably the best news for Java-based projects is that the most popular of these tools are free and open source.

The first step to building a solid CI pipeline is setting up a source code/ version control management process. Highly popular choices are distributed Version Control Systems (DVCS), such as Git and Mercurial. They allow developers to work offline and commit/revert changes locally before exposing any of their code into the shared repository.

The second step in the pipeline is to set up a Continuous Integration server that is employed to manage the entire process. Your CI server is going to

be your new best friend, because it is going to orchestrate the following and more for you:

• Poll the source repository for changes

• Execute automated builds

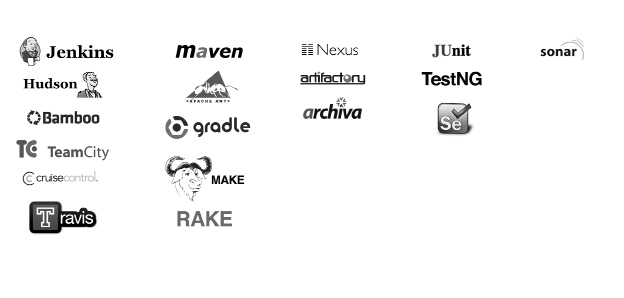
• Submits binaries to your artifact repository

• Run tests and code quality analysis,

• Notifies developers about failures in any of the phases in the pipeline.

**Investigate** these tools

**CI SERVERS: BUILD AUTOMATION TOOLS: ARTIFACT REPOSITORIES:**



**TEST FRAMEWORKS:**

**CODE ANALYSIS:**

The following diagram should provide a better overview and understanding of how the tools interact in CI ecosystem.

**Coming back to Darwin's Theory of Natural Selection, let’s try projecting it to the software world:**

**Would you agree that in the last decade we’ve seen that companies, especially start-ups, that were able to evolve and adopt state-of-the-art agile methodologies in software development & deployment (i.e. think Continuous Integration and Delivery/Deployment), felt themselves more capable of survival and indeed have shown a record of success?**

We hope the last few pages have been enough to convince you that if you aren’t playing around with your CI server regularly,

then you might be threatening your own survival! In the next section, we’ll give our top five reasons in support of incorporating CI

into your project.

**Five Obnoxiously Obvious Reasons Why**

Your Organization Should Implement CI

**HIGHER PROJECT QUALITY**

CI merges all developers’ code on a regular basis. This will ensure that your code, at the very least, compiles and leaves you without any broken builds. For multi-developer projects, this is highly recommended for achieving higher project quality.

**HIGHER CODE AND PRODUCT QUALITY**

Why not instruct your CI ecosystem to run quality controls on your project (yes, you can do that)? Your project’s health can be assessed by using methods like code coverage for tests, static code analysis for common bugs and pitfalls, measuring & profiling performance and automatically testing your app’s functionality.

**FIX PROBLEMS AS SOON AS POSSIBLE**

Who broke the build? Yep, things are going to go wrong. This happens often during software development. Maybe just an uncaught exception, where the solution isn’t difficult? Regardless, you want to know about these issues as quickly as possible. If your code gets merged multiple times each day and tests are run then you can achieve a very quick feedback cycle, which keeps you on your toes to fix something if it breaks.

**TOOL TO RELEASE SOFTWARE**

Releasing a software package that you can give into the hands of your testers, colleagues and clients is not an easy task. It can consist of very many steps that can and will go wrong when done manually. Automate this and make sure that your CI server knows how to release your software.

**CONFIDENCE TO RELEASE SOFTWARE**

When do you consider a software release to be “ready”? When QA gives the go ahead? It’s best to save your relationship with QA by not hitting them with every tiny version you push and getting a lot of bug reports. You need to have your tests green and you want to have a strong test suite that you can truly trust in order to release your software with confidence. Setting up your CI with a powerful testbed will help you get a good night’s sleep after a release.

***BONUS REASON - FROM DEV TO OPS AND BACK AGAIN***

*Ok, there is a sixth reason. Although created for Developers, Operations teams are using CI more and more these days. CI servers have evolved to the point that you can even orchestrate the deployment of your software. Are you writing a*

*web application that you also run in testing, QA and production environments? Let your shiny CI server handle the deployments as well. With a library of ever- evolving plugins, CI servers can now be used for almost anything that needs to be automated (however, walking your dog is still a couple years away).*

If you are at least partially convinced, then check out our next section, which shows you how to get started right away with 3 popular CI tools: Jenkins (created by Kohsuke Kawaguchi), Bamboo (Atlassian) and TeamCity (JetBrains).

**Jenkins**

Jenkins is a very popular free and open source CI server, started by a single developer (Kohsuke

Kawaguchi) over 7 years ago (under the name “Hudson” at that time). Jenkins is a solid, stable

and highly extendable piece of software, which many developers now consider to be the de facto standard for Continuous Integration servers.

So let’s try it out. The first thing we need to do is download and run Jenkins,



and then we’ll go a step further and add a project to be built by Jenkins.

Please follow these steps:

1. Go to Jenkins page (<http://jenkins-ci.org>).

2. Download the Java web archive or any native package. We will use the

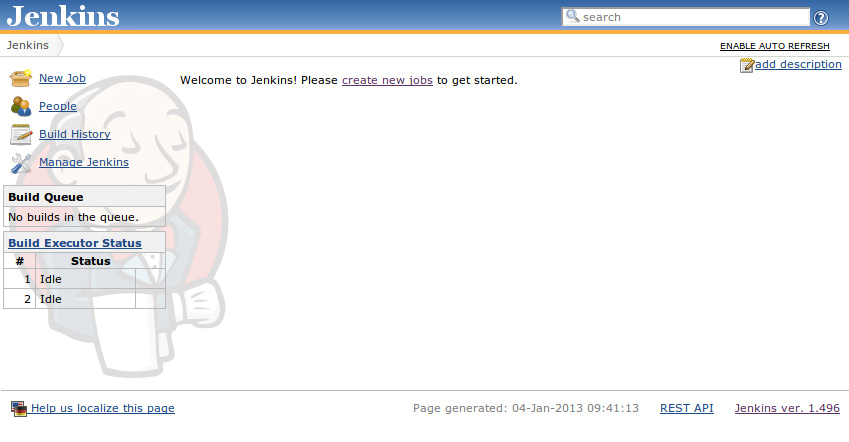
Java web archive in our how-to.

3. Start Jenkins by running "java -jar jenkins.war", if you downloaded war or

"start Jenkins service", if you chose native package.

4. Now navigate to <http://localhost:8080/>and you will see the main page of Jenkins.

Jenkins should now be running, so feel free to browse around a little bit and get acquainted. Now we will add a project for Jenkins to build.



Jenkins gives us hints for creating a new job, but before that we need to download some plugins. Bare Jenkins does not have much functionality, but it has a lot of plugins (>400) that make it much more powerful. We need to install the Git plugin to be able to clone from Git repositories.

1. Go to “Manage Jenkins” > “Manage Plugins”.

2. Switch to the “Available” tab and look for “Git Plugin”.

3. Tick the checkbox next to it, and press “Install without restart” button.

Jenkins will now download and install the



Git Plugin for you. Next, we’ll go back to the

Jenkins main page and create a new job.

For simplicity’s sake, we’ll name it “zt-zip” and select “Build a maven2/3 project”, then **click OK.** We will be transferred to a job configuration page.

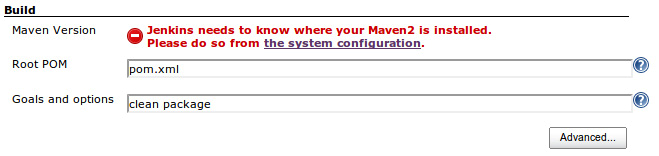


On the job configuration page, we must provide the Source Code Management (SCM) URL and build steps.

irst scroll down to Source Code Management.

**Select Git**, provide your repository URL and input branch as master. You can also add a repository browser, in our case it’s githubweb and the URL is the same as the repository URL. Now scroll down to Build steps.

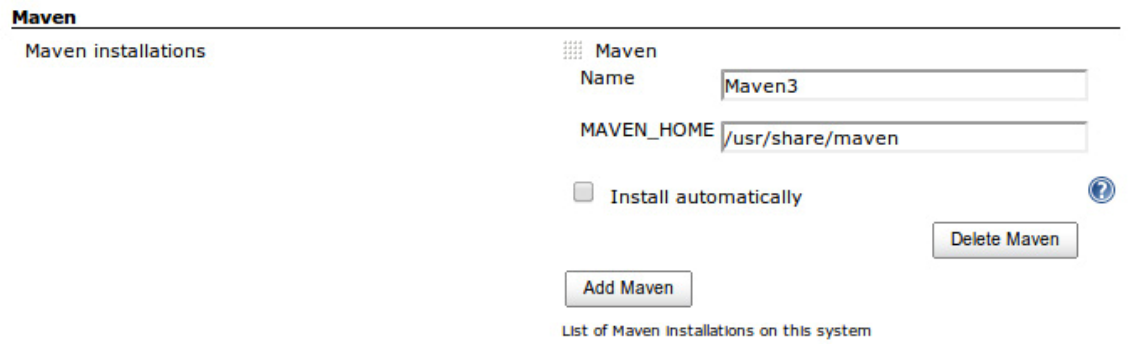
Now input “clean package” as a goal and press **Apply**. We now see a warning that Jenkins does not know where Maven is installed. We tell it by clicking the system configuration.



You will be redirected to a Jenkins configuration page. Scroll down to the “Maven” section and press **Add Maven**.

**Uncheck** “Install automatically checkbox. We have installed Maven through the package manager, so MAVEN\_ HOME is in */usr/share/maven*, but this name can be any string you like for easy reference later.

Click **Add Maven** and save the configuration. You will be

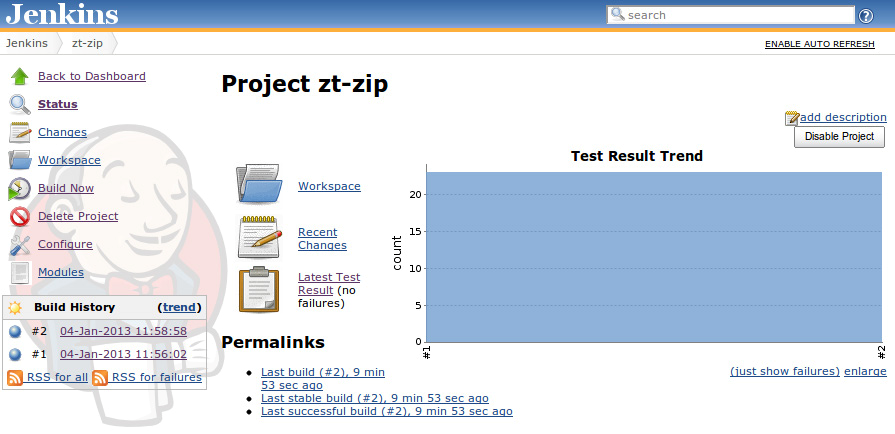


redirected back to the Jenkins main page.

Now we are ready to run the job. Press the **Schedule a build** button (the one with a green triangle and clock at the bottom of the page) and wait for the job to finish. Once the build finishes successfully, you can click on the name of the job

and go to its status page.

Here you can see build history, test results, some other info. To get the test result trend graph, you’ll have to run the job at least twice in order to get any stats.



**That’s it!** You have configured and run your first CI project

in Jenkins. You can improve this job by further configuration, such as setting up a process where Jenkins will send an email upon a failed build, run builds periodically, build on push to repository and so on