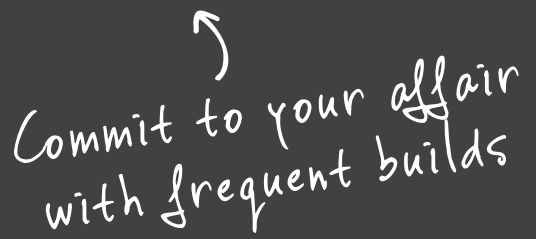
WHY DEVELOPERS  CI:

A GUIDE TO LOVING CONTINUOUS INTEGRATION



REBELLABS

PART I
INTRODUCTION

"A man who dares to waste one hour of time has not discovered the value of life."
-Charles Darwin, creator of the Theory of Evolution by Natural Selection

In most Java organizations, spending your time wisely when developing & deploying your apps to production is not only a major business priority, but an essential need to maintain existence in the future world of software.

Do developers care how they spend their time?

According to ZeroTurnaround's **Developer Productivity Report**, 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 RebelLabs 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 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:

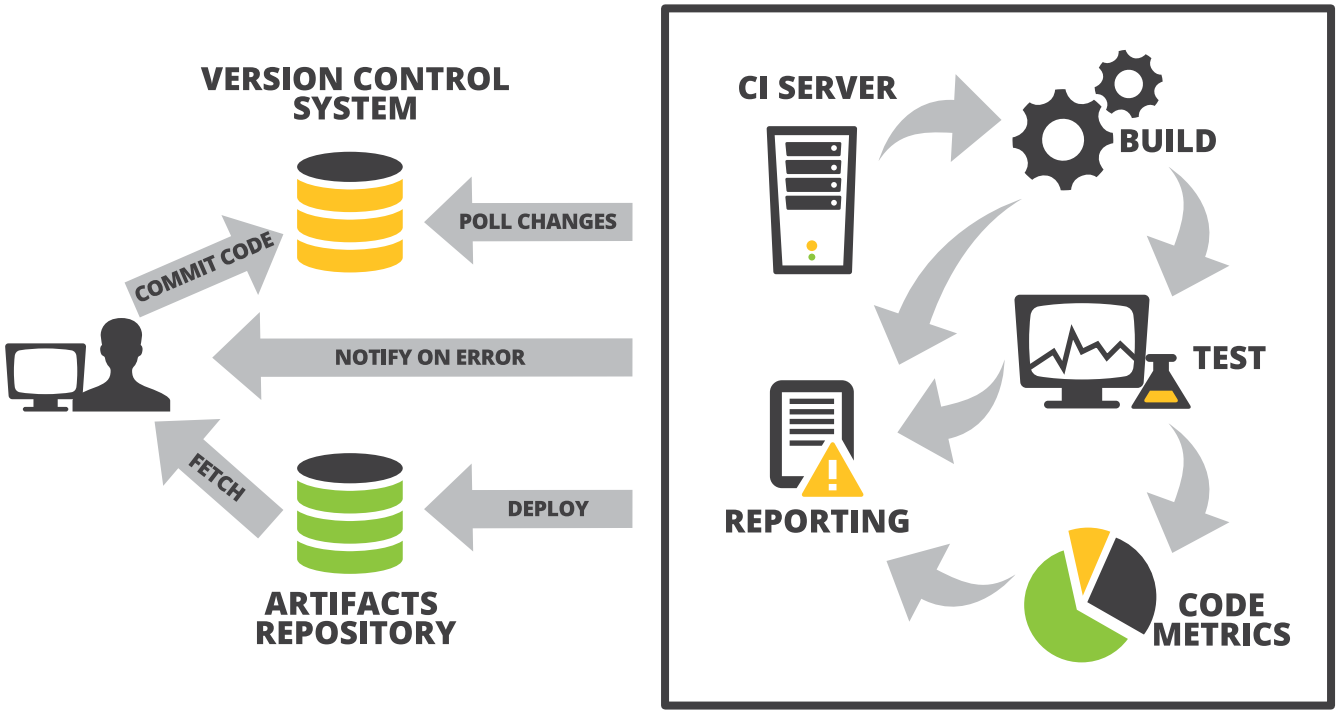
BUILD AUTOMATION TOOLS:

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).

PART II
GETTING STARTED WITH JENKINS, BAMBOO AND TEAMCITY

Now it's time to configure your first CI job - after this, you will have Continuous Integration enabled on your very own Project X!

Let's put your first project on to a Continuous Integration server: we'll review how to build it, run tests and publish test results. The demo project we are using is zt-zip (**https://github.com/zeroturnaround/zt-zip**). It is a Maven based open source project, so you can clone it and repeat all the steps exactly as we show here. You have to install the JDK, Maven and Git on your computer on your own before continuing. The CI servers we introduce are Jenkins, Bamboo and TeamCity.

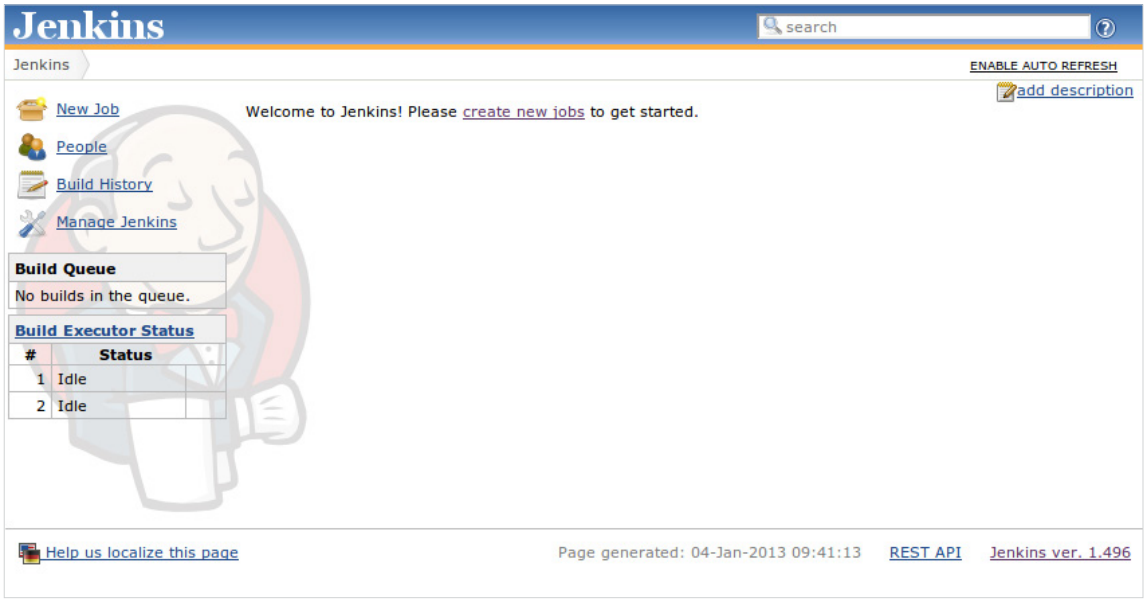
Jenkins

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.



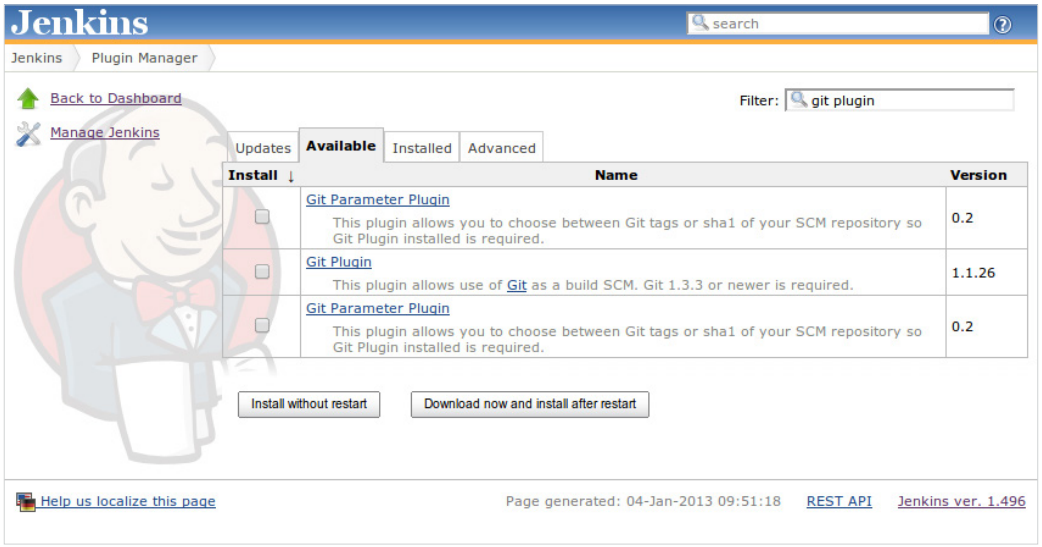
Jenkins should now be running, so feel free to browse around a little bit and get acquainted. Now we will add a project for Jenkins to build.

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”.

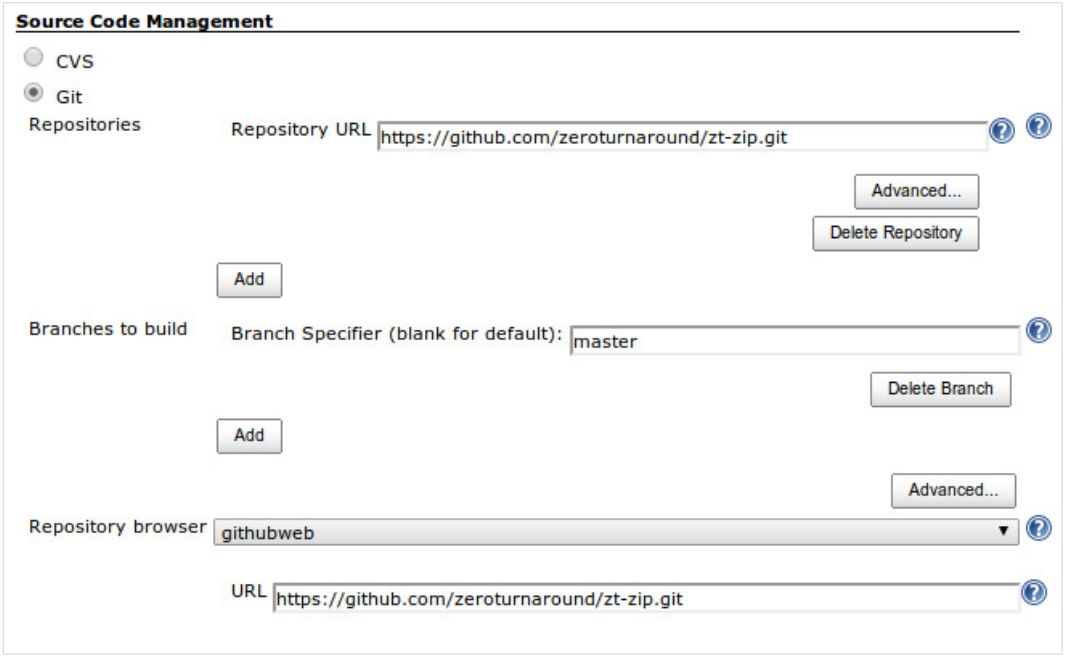
3. Tick the checkbox next to it, and press “Install without restart” button.



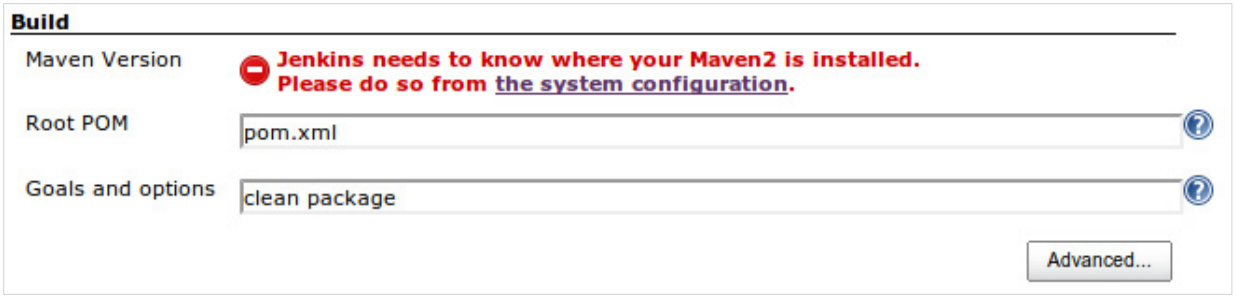
Jenkins will now download and install the Git Plugin for you. Next, we'll go back to the Jenkins main page and create a new job.

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.
First scroll down to Source Code Management.



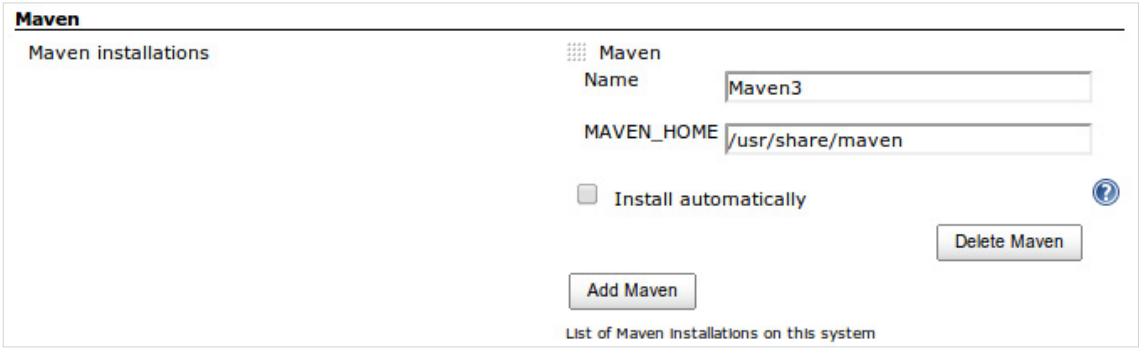
Select Git, provide your repository URL (https://github.com/zeroturnaround/zt-zip.git), and input branch as master. You can also add a repository browser, in our case it's githubweb and the URL is the same as the repository URL. Now scroll down to Build steps.



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**.

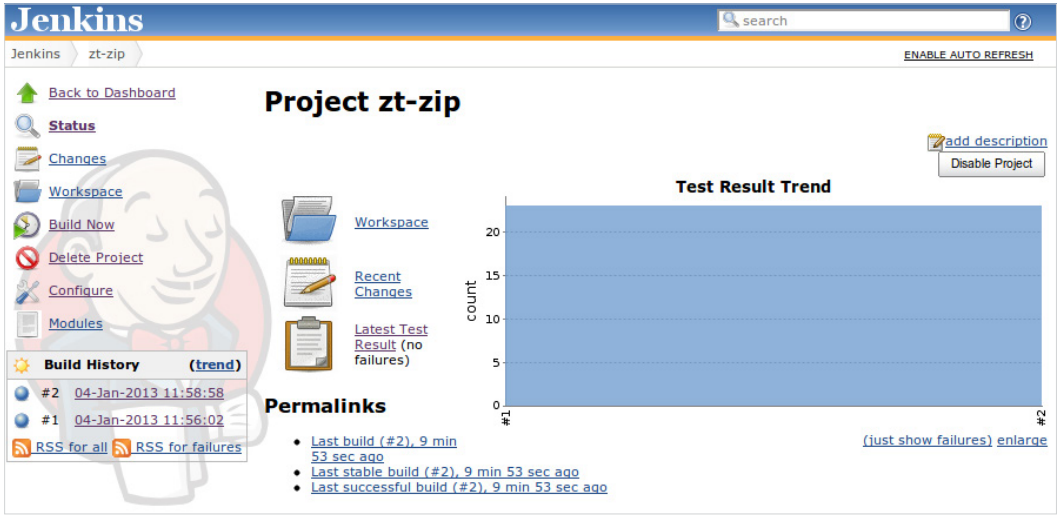
You will be redirected to a Jenkins configuration page. Scroll down to the “Maven” section and press **Add Maven**.

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.

Bamboo

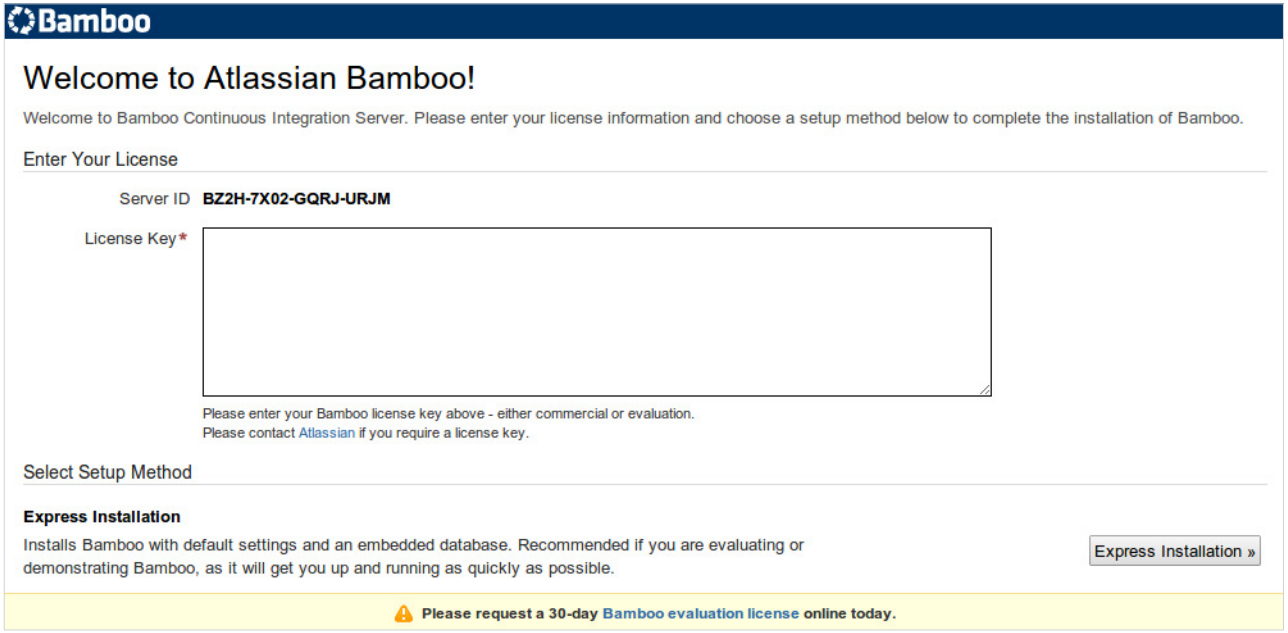
Bamboo is a commercial CI server from Atlassian. Unlike Jenkins, you have to pay for licensing, but you can try it with a free evaluation license.

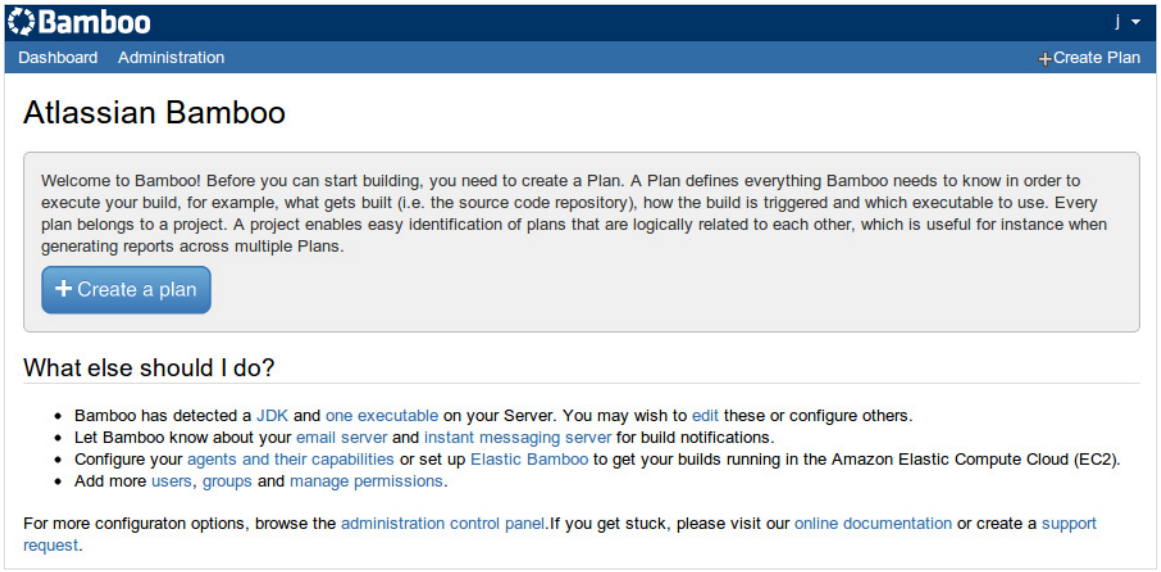
You can download it from http://www.atlassian.com/software/bamboo/overview. There are several installation possibilities for installing Bamboo, but we will take the "tar.gz archive". All we need to do then is to unpack it into the right directory.

1. Download your archive and unpack it into some directory (= "/opt" in our case)
2. Go to $bamboo_home (= "/opt/atlassian-bamboo-[version]" in our case)
3. Edit $bamboo_home/webapp/WEB-INF/classes/bamboo-init.properties
   set bamboo.home property (bamboo.home=/opt/bamboo-home)
4. Now start Bamboo by:

```
$bamboo_home/bamboo.sh start
```

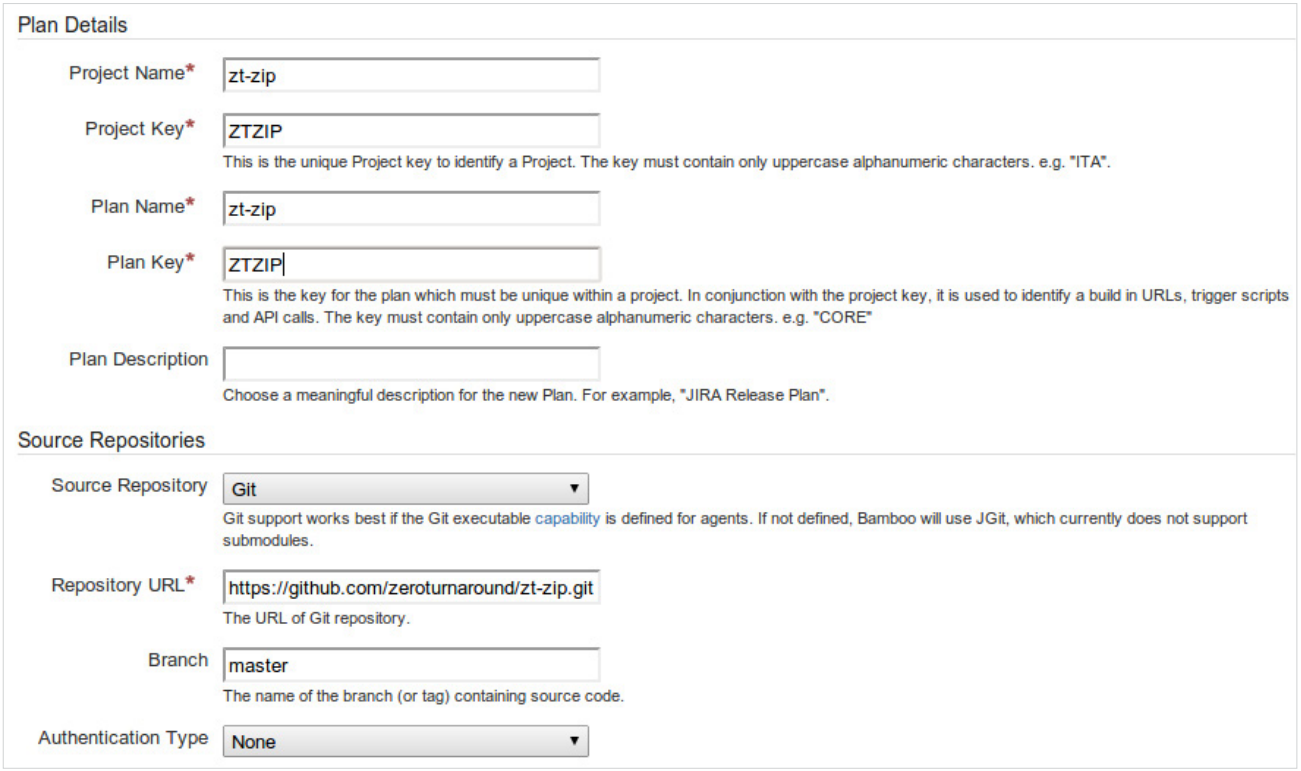
5. Now you can go to http://localhost:8085 and check if Bamboo is running.





You will see the welcome screen which will going to ask you for a license key. If you don't have one, you can opt for the evaluation license by clicking on **Bamboo evaluation license** at the bottom of the page. As soon as you get the license key and have finished the installation process, you will be redirected to the Bamboo main page.

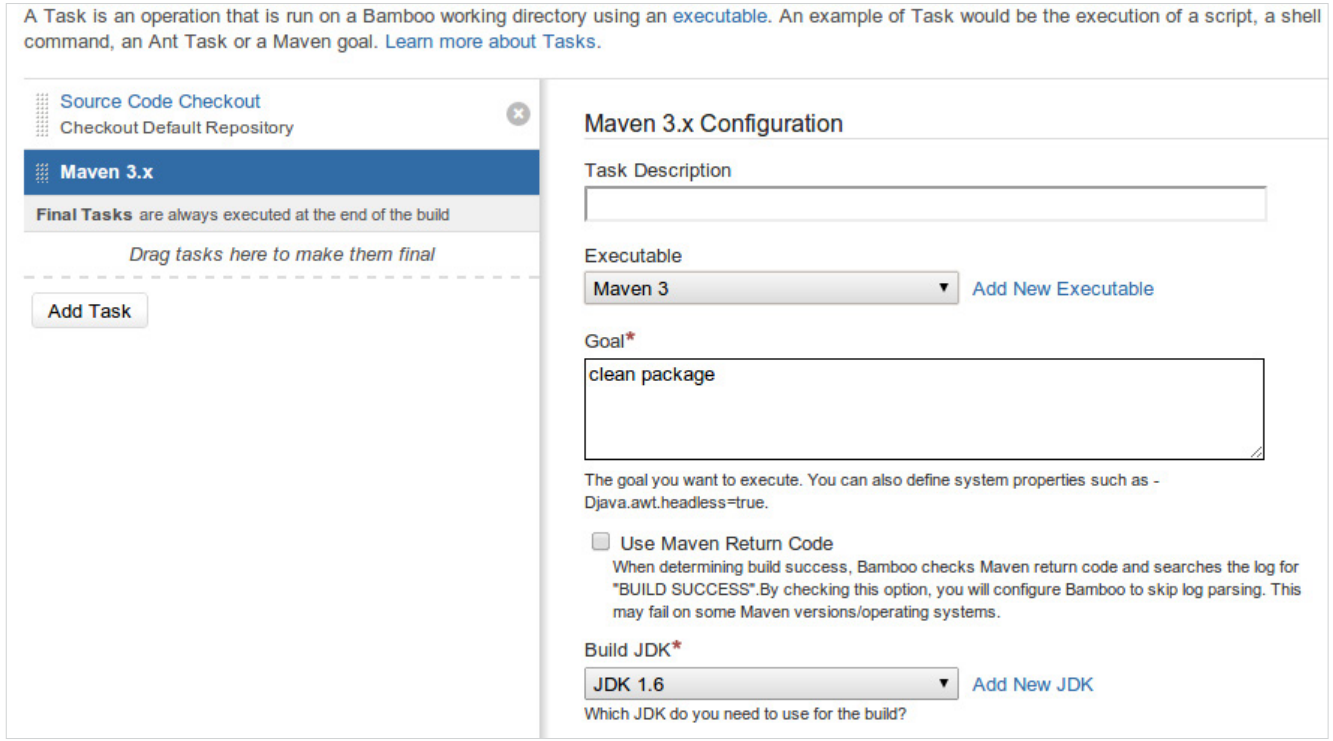
Why don't you browse around a little bit and get acquainted? The next step is to create a plan. Click **Create a plan**, then **Create a New Plan** in the newly-opened page and configure the new plan according to this screenshot:



Again, we've set the project and plan name to zt-zip, and the project and plan keys to ZTZIP (note: they must contain only UPPERCASE letters).

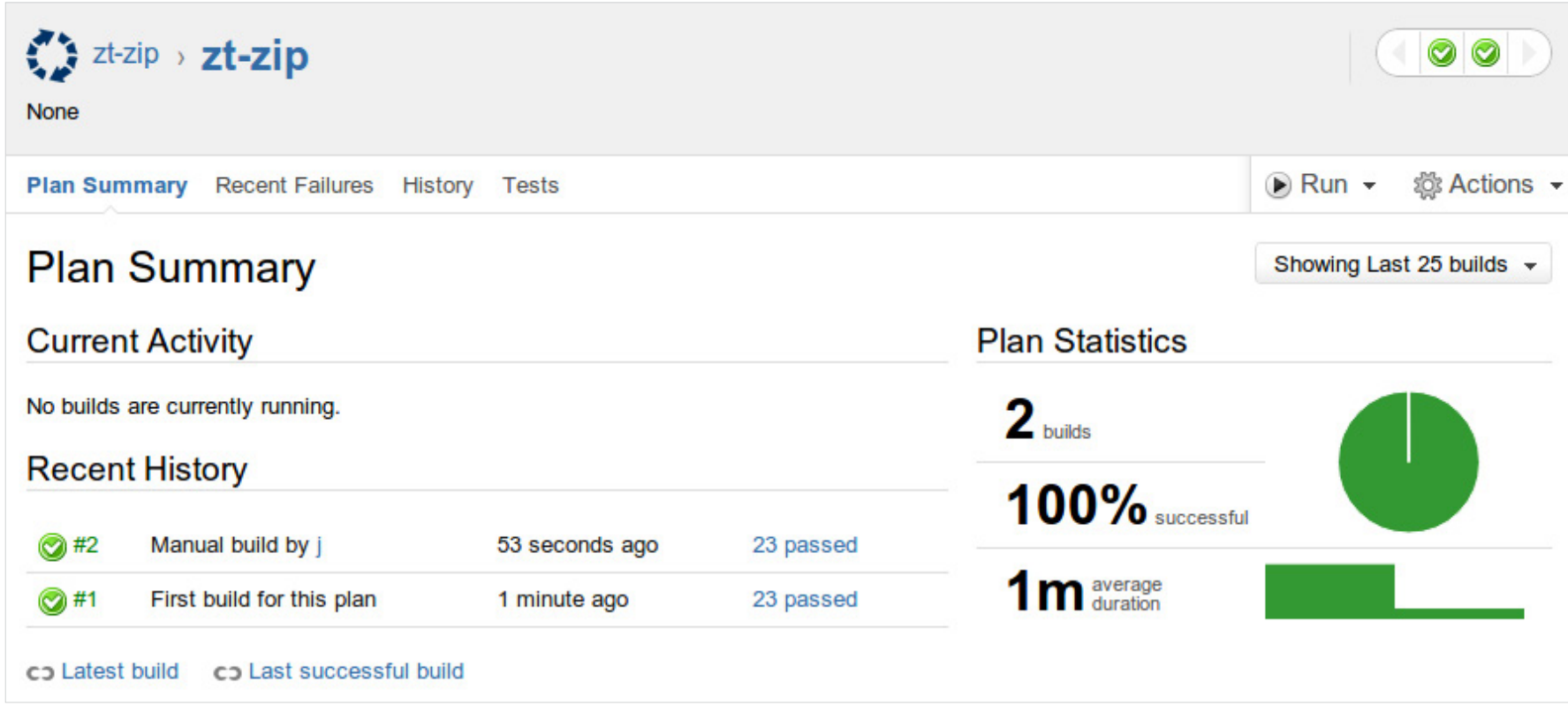
Select *Git* as the source repository and set the URL to https://github.com/zeroturnaround/zt-zip.git and branch to master.

Now we can continue onwards to configuring tasks for the plan. Add a task, select **Maven3.x** and set the goal to “clean package” (as in Jenkins). Bamboo should find your already-installed Java and Maven automatically.



We are almost there. Save the task, enable the plan and click Create. You are redirected to the main page of the plan, now you can run it a couple of times by pressing the Run button to see the results.

Now we can continue onwards to configuring tasks for the plan. Add a task, select **Maven3.x** and set the goal to “clean package” (as in Jenkins). Bamboo should find your already-installed Java and Maven automatically.



Here you can see the build history, test results, even average duration of the builds and some other stuff.

Congratulations! You have configured and run your first CI project in Bamboo.
Feel free to improve this plan on your own.

TeamCity

TeamCity is a CI server from JetBrains. By default it is provided with a free license that gives full access to all product features, with no time limit; the only restriction is a maximum of 20 build configurations. If you need more configurations, you have to pay for the product.

You can download it from http://www.jetbrains.com/teamcity/download/index.html. There is only a tar.gz archive available for Linux, so we will use it. Again, all we need to do is unpack it.

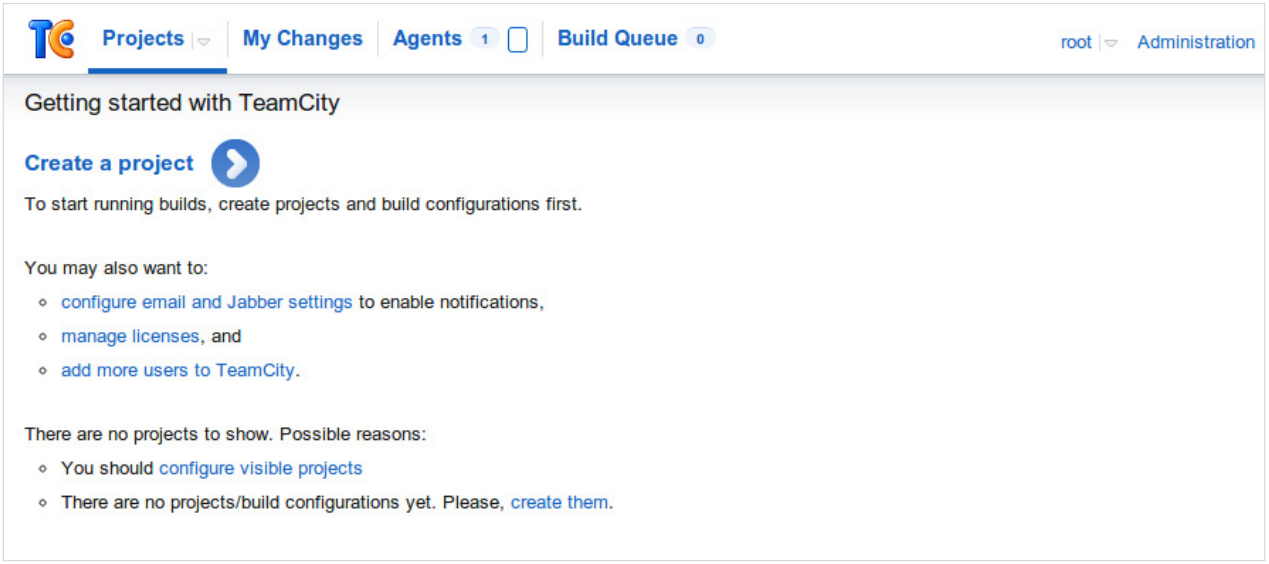
1. Download your archive and unpack it into some directory (="/opt" in our case)
   It creates /opt/TeamCity directory, which we will address as $teamcity_home

2. Go to $teamcity_home ('/opt/TeamCity')

3. Set TEAMCITY_DATA_PATH environment variable. That is the path where all the data about projects and builds in TeamCity will be saved. And start TeamCity:

```
TEAMCITY_DATA_PATH='/opt/TeamCityData'bin/runAll.sh start
```

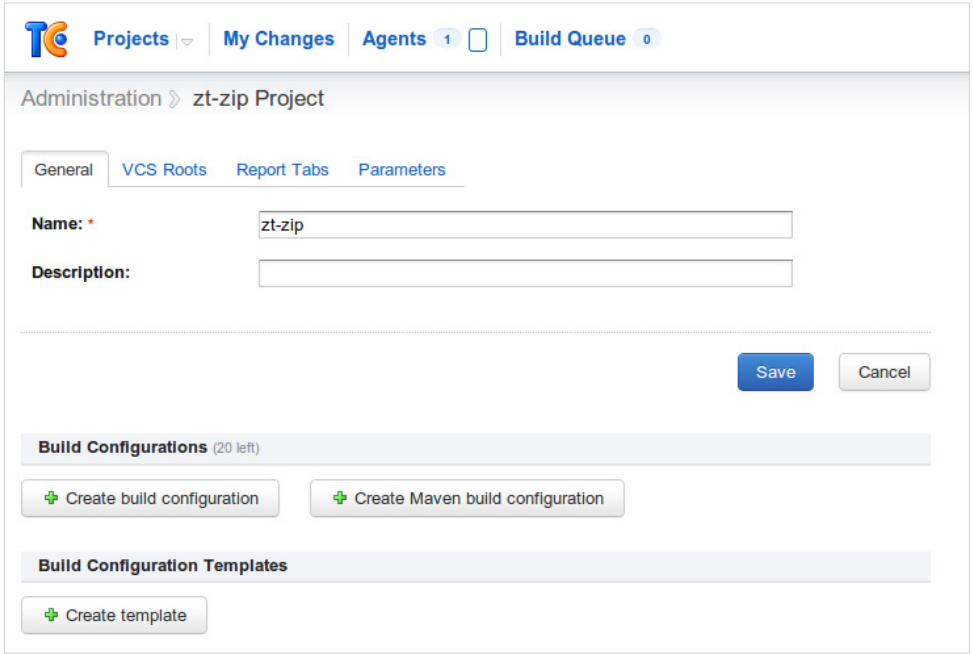
4. Now you can go to http://localhost:8111 and check it out.

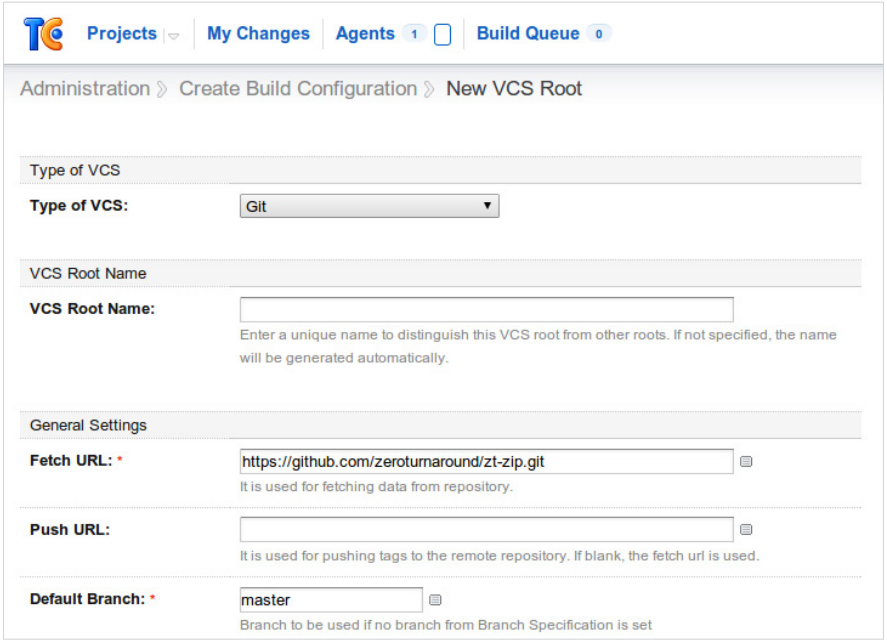


Once TeamCity has initialized and you have created an Administrator account, you get redirected to the main page.

You can browse a little bit around and get acquainted. Click Create a project when ready. Provide the project name once again (zt-zip) and then click Create. Now you are on the project configuration page.

We need to provide build steps. **Click Create build configuration**, give the new build configuration a name (zt-zip) and continue to VCS Settings.

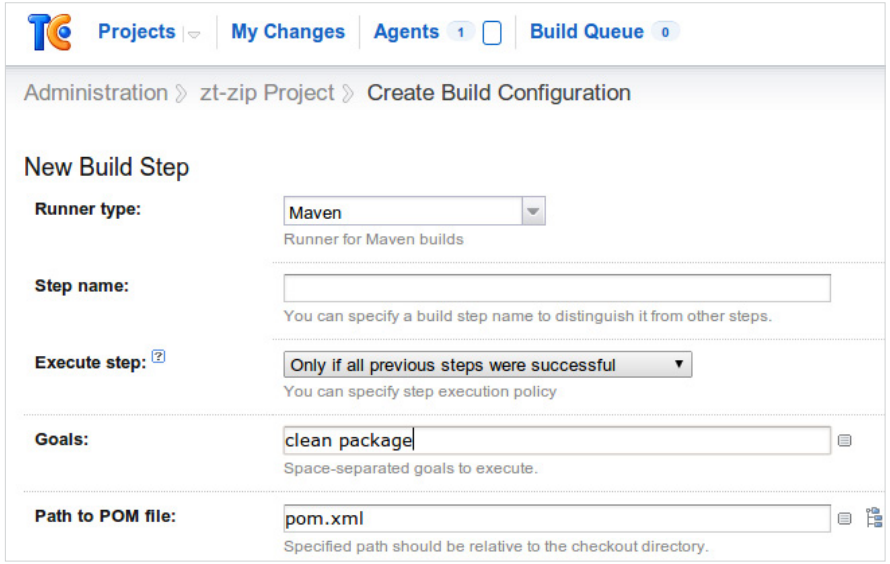




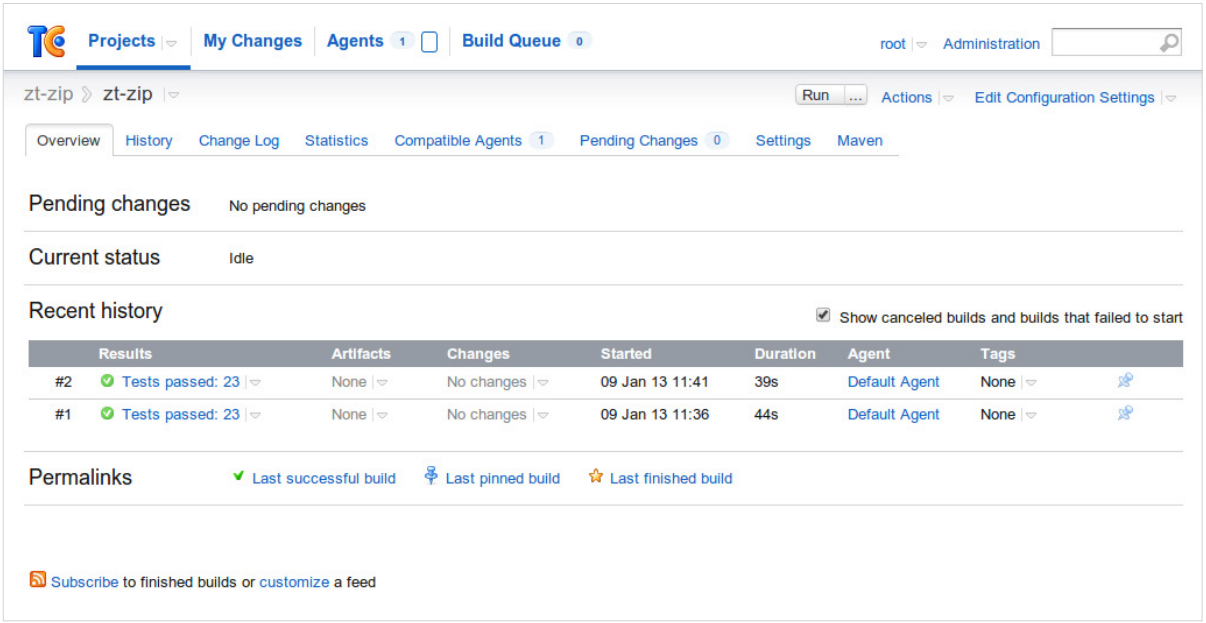
On the Version Control Settings page click **Create and attach new VCS root**. Select "Git" as type of VCS. Then provide "Fetch URL" (https://github.com/zeroturnaround/zt-zip.git) and Default Branch "master". You can test the connection before saving.

Save the VCS configuration and continue to the configuration of build steps. Select "Maven" as Runner type and set Goals to "clean package". Then save the configuration.

Now the project is ready to be run. Hit the **Run** button in the upper-right corner. While the project is being built, you can switch to the project page and wait to see some results.



Here you can see the build history, test results, duration of the builds and some other stuff.
You can switch among tabs or click on any build to see more detailed information.



There you have it. You have now configured and run your first CI project in TeamCity.
Feel free to improve this project on your own.

PART III
CONTINUOUS INTEGRATION INSIDE YOUR TEAM & ORGANIZATION

Whether you are trying to implement CI in a team of 3 or throughout a multinational organization of 30,000, you should keep in mind that there will be many changes coming in.

And we're not just talking about your software stack and hardware configurations. There are many social & even psychological considerations to take in....

Continuous Integration in a Team

Getting started with CI is easy for top-notch developers like you, but to take it to the next level can be difficult because this time it's not about software. It is a combination of software and people. Changing how people think and act is a lot more difficult than simply installing a software package.

To fully take advantage of CI, you need to have good tests running in your project. We recommend that you start off gently, with unit tests, which are great for testing the smallest unit of work. Of course, you will also want to incorporate functional tests, integration tests, smoke tests and others. But lets start with unit tests and not get overwhelmed.

Now that we've shown you how to configure a CI job and you've introduced the tool to your team, we're going to tell you that only half of the battle is won. The important things to do now is to keep your tests green and also make sure that developers use, respect and love your CI server.

Keeping Your Tests Green

First of all, you should make sure that when something fails then the right people will find out about it ASAP. The best solution is that if the person who broke the build is also identified and notified so that they can fix or at least debug the problem.

These notifications can be handled by email, as this is the most common way and supported by all of the CI servers. When we implemented our CI server at ZeroTurnaround, we found that emailing was not working for every team. So one development team wrote a Skype bot handler that SPAMs the entire team chat when something breaks. And let us just say that, WOW, this galvanizes developers into fixing stuff a lot quicker. You can check some other ways of doing this on the Jenkins wiki.

One fun solution that we like is literally wearing physical hats. In this case, each team has a test master who keeps an eye on all the tests. Once they see that somebody has broken the build, they takes out a special "Broke the Build!" hat and gives it to the developer responsible. This developer then needs to wear the hat until the issue gets fixed. This isn't such a viable solution for remote teams, but the social impact that this has will help you keep your tests green.

But if you want to make an impression, then go into full retaliation mode for broken builds - set up a cannon to shoot the person who broke the build!

> "When I'm deep into code, I find that emails are a distraction. Should developers be asked to keep an eye on their inbox more than couple of times a day? I'm sure managers would love that but for high productivity a developer shouldn't be reading email more than twice a day."

Toomas Römer,
Director of Engineering & Founder at ZeroTurnaround.

Love Your CI Server (Don't Ignore It!)

Another aspect of CI that you need to pay attention to is making sure the entire team is willing to adopt your new CI server, and making sure it's getting the love and attention it needs. New-to-the-game developers can be intimidated by the CI servers' apparent magic and feel more comfortable staying away--we recommend training at the junior level about CI to get that love fest going early.

But there can can be many reasons for ignoring your CI server, broken unit tests and learning to ignore broken builds, for example. The one we had to deal with at ZT is that the CI server was hidden in some remote location that was difficult to lookup. Simple barriers like this are enough to keep your relationship with your CI server quite cold.

But by using the powers of reverse proxies, cloud, cloud, ~~Lisp~~ and ~~little bit of Joel Spolsky~~ your infrastructure team can help you set it up in an easily accessible way. For example, using http://ci.company.com or internally http://ci.

Another reason for low adoption might be the **slow feedback** cycle. A developer is programming away, commits their code and only discovers at the end of the day that their morning commit broke the build. You are doing it wrong. The developer should get feedback a lot quicker, in an ideal world in less than 10 minutes. In reality 30 minutes is something quite easy to achieve (check the repository every 15 minutes and make sure your build and unit tests is less than 15 minutes).

It's certainly possible that your whole test cycle might take a very long time if the number of platforms is large (yes, we have the same problem), so it makes sense to break this up a little. We've even seen test cycles get even more messy because one developer does a second commit before the first build finishes! Make sure to define certain results to arrive just a couple of minutes after the build. This will give developers feedback faster and make them happier and more productive.

Once you have an automated notification process set up, another consideration is how to actually debug if something goes wrong. CI server installations in large enterprise teams and companies running many servers means that not everyone has system-level access to them. This is where your CI servers comes to help. For example, Jenkins lets you go through the workspace and view log files over HTTP, and email notifications also contains the build exception. You can even specify the files that you are especially interested in so that they get archived and have prominent links. This might even make debugging easier than if the developer does it on their own machine. In the end, what you get is a failure in a clean environment and meaningful logs to review so you don't do it again in production. Win!

What do we mean by a "clean environment"? Totally vanilla: an environment that is not custom-configured to pass tests, but simply a fresh machine with default configuration. So if the tests fail in this environment you will know that probably something is wrong with your application and not the environment.

Project Health

Once we have successfully introduced CI into the team, and you start actually taking advantage of it, then we can move a step further. Let's also add some software quality tools to the mix. We might want to measure the test code coverage when we run tests, so let's do some analysis with static code quality tools for smelly code, bugs and common pitfalls. It's fair to say that this topic deserves an article all of its own, but we will mention some tools that you might want to check out.

- **Sonar** - http://www.sonarsource.org/
  One system to rule them all. It will give you an overview of test results, code coverage, code metrics, coding rules compliance, time machine. It also integrates well with CI tools.

- **FindBugs, PMD, Cobertura or Emma, Checkstyle, JLint, JDepend, JavaNCSS, Jalopy** to name a few.
  Check out http://www.javacodegeeks.com/2012/10/java-code-quality-tools-overview.html for more information. Sonar has a good subset of these covered.

Figuring out the health of your project is just half of the battle. The other part is acting on it by defining your requirements and making this a reality. Here the only tip we suggest is to select a dedicated person whose job is to go over your results weekly or monthly, and in case of negative deviation, add bug tracker cases to fix those. This can be a role that rotates over a team of developers every few months, for example, but at any given one point there should be a go-to person.

Continuous Integration in an Organization

So far, we have covered the problems and solutions of introducing and using CI in a team. Now let's take a look at what happens if you have multiple teams and you want to implement CI in a larger organization.

The challenges of this are different and if you've gotten this far then it means that CI is considered a crucial part in the software development process of your organization. But the flip side of this coin is that if for some reason your CI server stops working then there might be a bit of panic going on.

CI Server Maintenance

We started this report by showing you how to get started with different CI servers. This was simply the introduction to CI, so for a production-level installation there are some other aspects that you should consider, like availability, scalability, security, backup solutions, monitoring and more. We won't go into details with these, but we will go through the CI specific ones.

SCALING

Once multiple teams are using the CI server, you'll find that you may quickly run out of computing resources. Developers and team leads will start complaining that software builds and tests are taking too long to execute because these tasks gets queued. Here the solution is horizontal scaling and all major CI servers let you set up simple agent machines to be managed by the master server. The master server will take a task from the queue and give this to a remote agent to execute (even in the cloud, ~~cloud~~, ~~cloud~~, ~~cloudy~~, ~~cloud~~).

The tough question about horizontal scaling is whether to do it in actual hardware, locally virtualized systems or to go to the cloud. Keep in mind, once you go into cloud you will have to start thinking about Infrastructure as a Service (IaaS) solutions and Platform as a Service solutions (PaaS) - and on top of all that, strong security!

THE QUESTION OF LOCAL HARDWARE

Local hardware has a lot of advantages that developers can appreciate, and until IaaS and PaaS start to take over the world, most Java organizations are probably going to stick with using local hardware for the near future.

> "We purchase desktop grade machines and we concentrate on number of executors and fast IO. Currently we purchase machines with i7-3930K CPU with 12 virtual cores, 16 GB RAM and 2 x 250GB SSD. We throw them to the server room, have a standard Ubuntu install, Chef will install necessary software and join the Jenkins Cloud. We name our nodes based on buildings from the game StarCraft so we are capped a little in terms of number of machines."

Toomas Römer,
Director of Engineering & Founder at ZeroTurnaround.

First of all, they are local which means that in terms of network latency, everything will be relatively fast. Also, dedicated, "bare metal" machines will let you customize the pure hardware to get you better performance for build times due to faster I/O. If you want to go with local hardware, then we suggest the following tips:

Make room for growth

Local hardware requires space, literally, to have room for servers, network appliances, power and cooling. Be sure to plan this ahead, because when you run out of room your hands are tied and either you find a bigger room and relocate all the servers (with downtime) or you have to go to the cloud.

Automate the adding of agents

When adding agent machines, be sure that you can do this in a quick manner. Network installations of the operating system and tools like Chef/Puppet for software installation and provisioning will help you a lot. Note: Dev and Ops teams differ as to who should set this up, each side tossing the ball to the other. This is where a dedicated CI guru might come in handy. Less bickering!

> "What is your magic number? Go back to monitoring and figure this out. How CPU, RAM and IO hungry are your jobs? Measure and then experiment!"

Toomas Römer,
Director of Engineering & Founder at ZeroTurnaround.

Manage all from a single location

Once you have multiple machines you don't want to upgrade software on each of them manually. Look into tools that will help you here, again we suggest Chef/Puppet as good options.

Performance monitoring

Although agent machines don't sound too important, monitoring is actually crucial if you want to have a well performing CI server cluster. CPU, RAM and I/O utilization is key here.

Executor count

CI servers have a notion of agent machines, previously mentioned, and they also have a notion of executors, or the number of concurrent jobs executing. In ZT, our experience taught us a good formula, number of executors = number of virtual cores - 2. So for 12 virtual core processor we give our CI agent 10 executors.

GOING WITH THE CLOUD

The cloud seems like a logical choice for CI because you can save on server costs by not running all the remote agents at the same time. With the elasticity that the cloud offers, you can scale when the need comes and do that within minutes (compare this to how long it takes you to purchase a machine and join the CI cluster?).

The three CI servers that we've mentioned above (Jenkins, Bamboo and TeamCity) all are cloud ready. For example, they have means to start and stop remote EC2 Amazon instances and run your remote jobs there. Of course, this means that you need to get the necessary data (e.g. source code for the build) to those servers and then download the results, which should be automatically taken care of - something that not all organizations are willing to do because of security reasons. If these "round trips" are not too messy and long, then why not give cloud a spin.

Another option for cloud is to choose a Platform as a Service (PaaS) provider. This means that instead of running dumb nodes on the cloud, you can move your whole CI into the cloud. Again, security policies in some organizations and industries make this a more difficult solution to adopt. One solution provider for this is CloudBees, who provides Jenkins in the Cloud.

> "From our personal experience we had to ditch the cloud approach because of the traffic between home and cloud. We did spend time to keep our EC2 images as up to date as possible but still the data transfer got too large and long."

Toomas Römer,
Director of Engineering & Founder at ZeroTurnaround.

SELECT A CI MASTER FOR YOUR ENTIRE ORGANIZATION

When multiple teams use the CI server, then it makes sense for somebody to be in charge of the health and future of it. At ZeroTurnaround, we learned that there has to be a dedicated person in the organization that manages your CI server software. This is not necessarily an infrastructure person, but more of a Dev/Ops-minded type that can fulfill the role of "CI Master" who will:

- Solve development team problems with the CI server
- Upgrade CI server software
- Install and upgrade CI server plugins
- Manage CI server configuration
- Give health checks - is the cluster being utilized or dying?

The CI Master is not a person who debugs your problematic builds, but they can certainly help if more access, debugging or CI help is needed.

In addition, it helps if every team has at least one person to interface with the CI Master who is more knowledgeable about the CI server. Then when they make configuration changes and exert any amount of control over the process, the CI master is more aware of any changes and can approve directly.

> "We call the CI master and his helpers the "Jenkins Overlords". On top of that our Jenkins cluster machines are buildings from StarCraft. We like Terran ones for Linux machines, Zerg for Windows and Protoss for Mac. It makes perfect sense."

Toomas Römer,
Director of Engineering & Founder at ZeroTurnaround.

CI SERVER AS ENTERPRISE CRON

We've talked about CI merging code, running tests and controlling the quality of your project. Can we take it a bit further? All CI tools have some sort of cron system, which is described by Wikipedia as:

> a time-based job scheduler found in Unix-like computer operating systems. Cron 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.

Compared to the cron we know, CI tools also add logging, cluster support, authentication, authorization and a GUI. Feel like getting a bit advanced already? Try using your CI system like a cron to automate operations that are out of the scope of Continuous Integration, like releasing your software, updating your website, deploying applications, analyzing statistics etc.

Coming back to review

Whew - that was quite a lot all at once. Let's not beat a dead horse - here are the main things we want you to keep in mind here:

BE SMART AND MODERN

It's no secret: If you want to be agile and productive in the software game, employ Continuous Integration methodologies so that after each commit to a shared Version Control System your app or system is automatically built and tested.

CI ADOPTION STILL NOT AT PEAK LEVELS

Lots of development teams still haven't found the key to deploying even free CI tools like Jenkins and Hudson, even though developers value their time and like to eliminate headaches from broken builds, manual merging hell and regressions.

MANY OBVIOUS BENEFITS - HIGHER QUALITY IN YOUR FACE!

With CI, you get higher project quality, higher code and product quality, the ability to fix bugs and issues as quickly as possible, a greater confidence in releasing new software versions, plus the tools to automate the release of your software when it's ready. Plus, it's pure DevOps too!

QUICKLY GETTING STARTED HERE

This report will guide you to getting started right away with Jenkins, Bamboo and TeamCity - other tools are recommended to look into as well.

SOCIAL CHALLENGES, YOU WILL CONFRONT

There are challenges other than just software to implementing CI processes in your team or organization. Social issues, such as getting the team to buy into the concept of CI, respecting your CI server and who makes sure all the tests are green, all run parallel to technological issues like scaling, maintenance and local HW vs. trying the cloud that need to be identified and considered before jumping in too deep.

IN THE END: FASTER DELIVERY OF BETTER QUALITY CODE

So, using CI, 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. 'Nuff said?

Are you writing software and not using CI? If so, please fix this and install something. Your boss, clients and even your pet will thank you.

P.S. in our next report, we dive deeper into Jenkins, the most popular CI server out there, and have an extended interview with our friend Kohsuke Kawaguchi, the founder.

RebelLabs is the research & content division of ZeroTurnaround

Contact Us

Twitter: @RebelLabs
Web: http://zeroturnaround.com/rebellabs
Email: labs@zeroturnaround.com

Estonia
Ülikooli 2, 5th floor
Tartu, Estonia, 51003
Phone: +372 740 4533

USA
545 Boylston St., 4th flr.
Boston, MA, USA, 02116
Phone: 1(857)277-1199

This report is brought to you by:
Toomas Römer, Juri Timošin, Pavel Grigorenko, Kostis Kapelonis, Ryan St. James and Oliver White