MIAGE

# LES ENJEUX DE LA MISE EN PLACE DE L'INTÉGRATION CONTINUE DANS DE PROJETS DE DÉVELOPPEMENT EXISTANTS

25 avril 2018

Francisco Javier Martínez Lago
Université Paris-Dauphine

# *Table des matières*

Chapitre One

# *Introduction*

The quest to maximize efficiency is an ever-present problematic in the entreprise. Efficiency can lead to a company to succeed while their competitor fails. It can be the difference between meeting a deadline and delivering a project late. It's no wonder that there are many articles and research papers dedicated to the topic.

In software development efficiency is often sought through automation. Removing repetitive, manual processes minimizes human errors and allows developers to focus their energies on more important tasks. However, the implementation of automation is not always trivial.

There are two cases where we can ask ourselves the question of whether to implement automatisation in our project. First, when we start a new project. Second, when we have an existing project that we wish to make more efficient by automation. In this document we will be examining the second case, specifically the difficulties involved in such an undertaking and some examples of real-world implementations.

Chapitre Two

# *What is Continuous Integration ?*

The term "Continuous Integration" was first used by Grady Booch to describe a technique of developing object-oriented software that involves each developer making frequent, incremental integrations rather there being single a "big-bang" integration event. This strategy was meant to reduce developer risk by finding issues early in the development process[1].

? ? ? https ://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration

The term has evolved over the years and is now understood to be the process of automating building, testing and validating code every time a team member commits changes to a shared version control repository. This encourages developers to share their changes as soon as they finish a small task and helps find issues quickly. Committing code to the master (or main) branch triggers an automated build system to grab the latest code, build it, test it and validate it, all automatically.

Continuous Integration is the answer to the issue of developers working in isolation. When they take too much time to integrate their changes into the main branch there can be issues such as merge conflicts, diverging code strategies, duplicate efforts and bugs. Because CI requires the team to integrate their changes continuously to a shared branch, it helps mitigate these problems. [1]

1. https ://www.visualstudio.com/learn/what-is-continuous-integration/

Chapitre Three

# *Different CI platforms*

There exist numerous platforms offering continuous integration services. In this section we are going to explore some of the main offering currently available on the market.

## 3.1 Jenkins

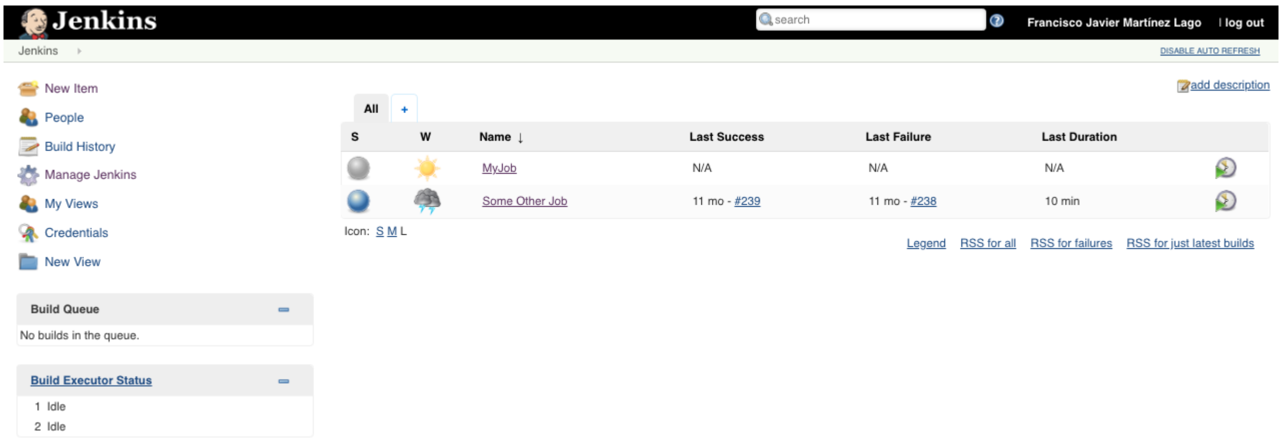
Jenkins is an open-source (OSS) CI platform whose initial objective has been the automation of the build and test processes. Jenkins runs as a Java server and is deployed through a .war file. It supports multiple version control systems and can execute Maven and Ant projects as well as shell scrips, Windows batch commands and many others. Jenkins functionality can be extended through plugins.

A basic workflow of Jenkins might look like this :

1. A developer submits their code.
2. Jenkins picks up the changes and triggers a build and runs the tests.
3. The output of the build will be available on the Jenkins dashboard. Notifications are sent to the developer.

To get anything done in Jenkins you need to create a job. A job consists of a source of data, such as a version control repository or a local directory, and a pipeline, which is basically a series of steps which will be executed when the build runs.

You also need to specify a trigger. That is to say, a condition upon which Jenkins will start running your job. The two main options are time-based (ex : every 20 minutes) and event-based (ex : every time there's a commit on your main branch), but there are also others. With all of this configured, your job will appear on the Jenkins dashboard.



| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|---|---|---|---|---|---|
| | | MyJob | N/A | N/A | N/A |
| | | Some Other Job | 11 mo - #239 | 11 mo - #238 | 10 min |

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

**Figure 3.1 –** Interface principale de Jenkins

The main interface of Jenkins, which is accessible through a web browser, is shown above. On the left there is a list of actions available such as adding new jobs or configuring Jenkins settings. Under that we can see the build queue. If there were any active builds that's where we would see them. Further down there's the build executor status. This has to do with advanced Jenkins functionality that allows to run Jenkins in a distributed environment.

Shown to the right is the job list. The columns, in order from left to right : Status, Weather, Name, Last Success, Last Failure, Last Duration. There's also an icon on the far right that allows to rerun the build manually.

**Status :** shows the status of the build. A build can either be successful, pending, aborted or failed.

**Weather :** it represents the general health of the build by a weather icon. The more your builds fail, the worse the weather gets.

**Name :** the name of the job.

**Last Success :** date of the last successful build.

**Last Failure :** date of the last failed build.

**Last duration :** duration of the last successful build.

## 3.2 Travis-CI

Travis-CI is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. Travis-CI continuously monitors changes in a version control repository, automatically builds and tests code changes and provides immediate feedback on the success or failure of the build [1]. Builds can be customized by using a special file called « .travis.yml »

A basic Travis-CI workflow might look like this :

1. A developer submits their code
2. Travis-CI picks up the changes and reads the travis.yml for project-specific settings.

1. https ://docs.travis-ci.com/user/for-beginners/

3. Travis-CI clones the version control repository into a new virtual environment and build and tests the code
4. Build succeeds or fails. Notifications are sent to the relevant developers.

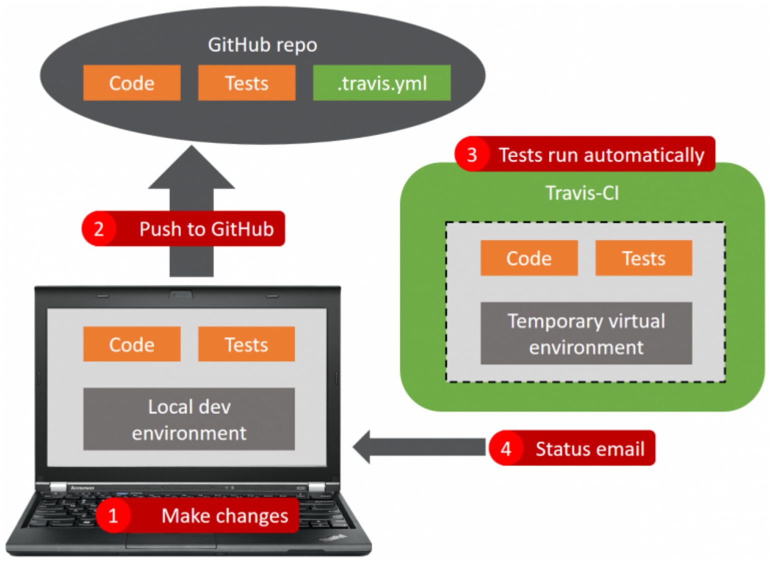


**Figure 3.2 –** Architecture de Travis

In Travis-CI there are some fundamental concepts [2] :

- **job** : an automated process that clones your repository into a virtual environment and then carries out a series of phases such as compiling your code, running tests, etc. A job fails if the return code of the script phase is non zero.
- **phase** : the sequential steps of a job. For example, the install phase, comes before the script phase, which comes before the optional deploy phase.
- **build** : a group of jobs. For example, a build might have two jobs, each of which tests a project with a different version of a programming language. A build finishes when all of its jobs are finished.
- **stage** : a group of jobs that run in parallel as part of sequential build process composed of multiple stages.

## 3.3 BuildBot

Buildbot is a job scheduling system released under the GPL license. It queues jobs, executes the jobs when the required resources are available, and reports the results [3].

A BuildBot installation consists of one or more masters and a collection of workers. The master periodically checks for changes in the source code of your version control repository, coordinates the activities of the workers and send the results of the jobs back to the developers. The workers can run on a variety of platforms.

2. Voir note 2 de bas de page
3. https ://buildbot.net/

BuildBot is configured by a Python configuration script on the master.
This is how a typical BuildBot workflow might look like :

1. A master machine polls the version control repository and pulls the latest changes
2. The master machine sends commands to the worker machines to build and test the code
3. The master receives the results of the jobs and sends the reports back to the developers through email.

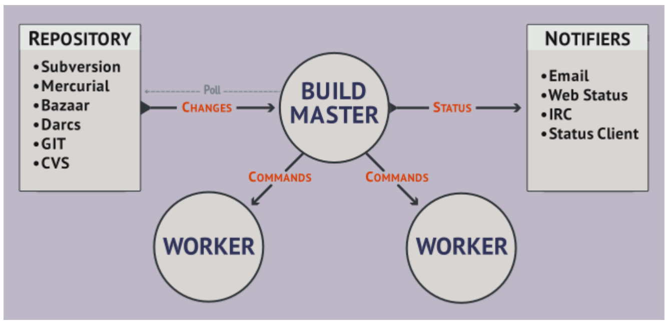


**Figure 3.3 –** Architecture de BuildBot

The workers typically run on separate machines, which can correspond each to a platform of interest. They send a request to the build master to initiate a TCP connection. If the connection succeeds, the build master uses it to send command and the worker uses it to send results of commands.

The worker needs to be able to access the version control repository as it doesn't get the source code from the build master.

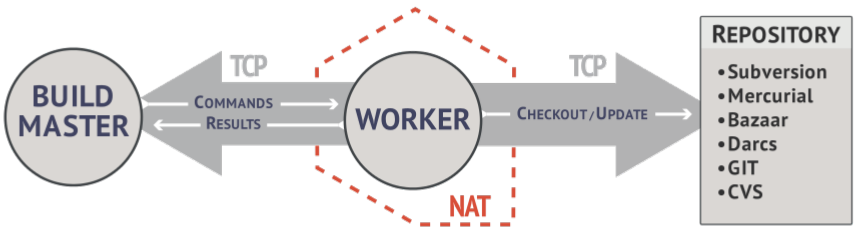


**Figure 3.4 –** Architecture Worker de BuildBot

? ? ? https ://docs.buildbot.net/current/manual/introduction.html

Chapitre Four

# *Integrating CI Into An Existing Project*

There are two scenarios that can take place when thinking of implementing CI into a project.

The first case is integrating CI into a completely new project. This is the most flexible stage as no code has been committed and there is room for implementing new ideas before goals are fixed.

The second option involves implementing continuous integration into an existing project. By an existing project I mean one which has gone from the conception stage to the development phase. Indeed, the direction of the project has been set and the development is in progress.

In this section I am going to explore the second option.

Integrating CI into an existing infrastructure could prove to be a non-trivial undertaking. Indeed, one need not only consider the the work involved in setting up the chosen platform but also the coding culture in place in the development environment. Perhaps the existing developers are accustomed to a cer

## 4.1 Phases dans un projet de développement

Let's take a look at the possibles phases that we can encounter in a project [2] :

### 4.1.1 Pas de serveur de build

There isn't a central build server. Builds are run on the developers' machines, either manually or through some script like Ant. The source code might be stored in a version control repository but the developers are not accustomed to regularly commiting changes to the code. Before shipping, a developer integrates changes manually.

### 4.1.2 Builds quotidiens

Now there is a web server which runs build regularly (usually at night). The server compiles the code, but there is no automated testing. Even if there are tests, they aren't part of the build process. Developers are commiting changes more regularly now, even if it's just at the end of the day. If there are code conflicts, the build server will notify the team in the morning.

### 4.1.3 Builds quotidiens et tests automatisés basiques

The team takes continuous integraiton a bit more seriously now. The build server now runs every time there's a code on the version control repository. The build script not only compiles but also runs automated unit and/or integration tests. The build server also sends instant notifications in case of a problem, not just by email but also through instant messaging. Failing builds are dealth with quickly by the team.

### 4.1.4 Arrivée des métriques

Automated quality metrics are now used to help evaluate the code's quality and the test coverage. The build measuring the code quality also created the documentation of the API ofthe application. There's also a dashboard available for all team members to see the state of the project at a glance.

### 4.1.5 Prendre les tests au sérieux

More thorough tests are performed after the basic tests are done, such as end-to-end tests and performance tests.

### 4.1.6 Tests d'acceptance automatisés et un déploiement plus automatisé

Acceptance Test Driven Development is used throughout the project to push development forward and to generate high-level reports on the state of the project.

### 4.1.7 Déploiement Continu

Trust in the continuous integration system is such that team is confident about letting modifications automatically be deployed to production.

This list is not exhaustive and the progression between the phases might not correspond exactly to real world situations, but it gives a global idea on the continuous integration strategy to implement in a real world organization.

???

Chapitre Five

# *Continuous integration in monolithic applications*

In this section we are going to take a look at the unique challenges of implementing continuous integration processes in monolithic application and some possible solutions.

## 5.1 What's a monolithic application ?

A monolith is basically a big block of stone. Similarly, a monolithic application is a piece of software built without modularity, a big block of code that's supposed to handle many different requests by itself.

Monolithic applications are usually big in size and scope. In addition, they are not easy to debug. Because the different components are tightly coupled, a change in one part of the code could easily have repercussions elsewhere.

## 5.2 Some tried solutions to monolithic issues

Many companies have had to deal with the issue of implementing continuous integration in monolithic applications. In this section we are going to see some of the solutions they have come up with.

### 5.2.1 Nextdoor Engineering [1]

Nextdoor is a a services tech company. They had a team of more than seventy engineers working on a Django monolithic application. They wanted to streamline their continuous deployment work flow, but they were facing the problem that their builds would take around 25 minutes each to run. If each build consisted of only one change, that means that builds would queue up and could take hours until they landed on production.

To solve this issue, they introduced the concept of a train. A train is simply a batch of releasable changes. To determine what's on a train, they have a simple logical process. When a change lands on the master, if there is already an existing train, it is queued for the next train. If no such train exist, a new one is created from the queue of waiting changes.

1. https ://engblog.nextdoor.com/3-hard-lessons-from-scaling-continuous-deployment-to-a-monolith-with-70-engineers-99fb6dfe3c38

Trains go through three phases : delivery, verification and deployment.

1. First phase : delivery : this includes the build and deployment of the changes to the staging environment.
2. Second phase : verification : this includes automated verification such as tests and human verification with JIRA.
3. Deployment : automatic deployment to production

The train functionality has allowed Nextdoor to group several changes together, which in turn solves the issue of single-change builds slowing down the whole continuous integration system.

### 5.2.2 Conductor [2]

Searchlight is a platform to get info about customers' search habits and deliver personalized recommendations. Because the application itself was built as a large, monolithic application, they were finding issues. They found that when a developer changed one part of the application they needed to test the whole application to ensure everything was working. In addition, if they wanted to improve performance in one area, they were forced to consider the application as a whole.

They decided to approach the problem by using microservices.

Ils ont choisi de construire de nouvelles fonctionnalités en tant que microservices. En outre, ils ont examiné leur application monolithique et ont construit certaines des caractéristiques existantes en tant que microservices là où c'était logique. For this new microservice infrastructure they decided to use Jenkins.

With this new infrastructure in place, they were able to reduce build and test times in order to deploy monolith code daily and deployment of microservices multiple times per day.

2. https ://www.conductor.com/nightlight/revamping-continuous-integration-delivery-conductor/

Chapitre Six

# *CI in software development methodologies*

A software development methodology or system development methodology in software engineering is a framework that is used to structure, plan, and control the process of developing an information system[1].

Nowadays in the enterprise there are many different methodologies in place for software development. We hear terms like Agility, Scrum, Kanban, XP, Lean Developemnt and more. Some of these, like Extreme Programming, are naturally suited for a continuous integration work flow, while others, like Waterfall, not so much. In this section you will find a small overview of a few methodologies and the challenges involved in using this frameworks with continuous integration.

## 6.1 Agile

Agile is an umbrella term that encompasses a series of methods and practices based on the values and principles expressed in the Agile Manifesto[2]. These values and principles are present in a number of software development methodologies such as Scrum, Extreme Programming, Unified Process or Evo.

The agile manifesto doesn't mention continuous integration directly. However, the majority of principles can be achieved partially or totally through it. Here are the five principles that are pertinent to continuous integration[3] :

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
- "Working software is the primary measure of progress."
- "Simplicity–the art of maximizing the amount of work not done–is essential."

Why does continuous integration work seamlessly with Agile ? Because it's a prerequisite to continuous delivery, it ensures working software by catching errors early and notifying the relevant developers and it minimizes the time developers need to spend on building and testing software themselves.

1. http ://www.itinfo.am/eng/software-development-methodologies/
2. https ://www.agilealliance.org/agile101/
3. https ://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

## 6.2 Waterfall

Waterfall is a software development model consisting of a series of well-defined steps which are as follow [4] :

1. Requirement Analysis : determine and understand the requirements.
2. System Design : create the design.
3. Implementation : create the program in units which are integrated in the next phase. There are unit tests for each of these units.
4. System Testing : integrate the code from the previous phase and do tests on the resulting code.
5. System Deployment : deploy the app.
6. System Maintenance : maintain the application and fix issues.

The project goes through each of the phases in order. When it's the implementation phase we only create the code and when it's the system testing phase we only do the tests. And there's no need to have a deliverable piece of software at the end of certain periods like in agile. Rather, the final application will be available on the system deployment phase.

Continuous integration accelerates software development by encouraging developers to integrate often. The waterfall model has no need of frequent integrations as the software specifications are well known in advance. Developers simply code what is written down in the requirements at their own pace.

Contrary to agile methodology, there's no need to have a deliverable piece of software at the end of certain periods. Therefore, making sure that every single build passes during the implementation phase is not as priority. What matters is that the software units meet the requirements at the end of the phase.

And finally, testing is done solely on the System Testing phase. There is hardly a need to implement automated testing across the board when there's only one phase where it takes place.

"In contrast, waterfall projects will tend not to invest in automation as a priority, primarily due to the phased investment and handoff mindset that accompanies that approach. At best, builds may leverage some automation, and a phase of a waterfall project may be dedicated to functional test automation for long-term support/future release needs, but it's rare to find much more in a back-end-loaded waterfall project" [5].

4. http ://www.softwaretestinghelp.com/what-is-sdlc-waterfall-model/
5. https ://smartbear.com/learn/automated-testing/testing-in-agile-environments/

Chapitre Seven

# *Challenges of migrating to continuous integration*

There are many aspects to be considered when looking at where to put in place a continuous integration infrastructure. In this section we are going to take a look at some of them.

## 7.1 Mindset

There are at least three issues that can stymie the implementation of continuous integration in a software project : skepticism, changing old habits and exposing work intention [3] :

**Skepticism.** For a team that has never worked with CI, the benefits of implementing such a practice might not be immediately obvious. They might question why they need to spend their time and effort implementing a new system when the old one works just fine. It's important to be able to clearly explain the benefits of integrating CI and to respond to the questions from the developers.

**Changing old habits.** Continuous integration requires developers to integrate frequently. If a developer is accustomed to developing code in isolation for a long time, he might find it hard to change the way he works.

**Exposing work intention.** Continuous integration requires developers to integrate early. Developers who are accustomed to integrating in a big-bang event might prefer to working on their branch for a while, fixing any bugs that might come up. With continuous integration, they are required to publish their work early, whether the code is high quality or not. This can be discouraging as a developer might feel anxious about his code being examined by his peers.

## 7.2 Understanding

Clear communication between the team and management is crucial to maximize the chances of a successful projects. Let's see a few of the issues that can hinder that :

**Unclear goals.** A possible way for an enterprise to introduce CI to the company is to have a pilot team do it first [1]. This pilot team can help the rest of the teams implement the process. Therefore, these other teams implementing CI could complain that there isn't a clear goal.

1. https ://www.cmcrossroads.com/article/how-implement-continuous-integration

Instead of this sort of freedom easing the life of the developers, it can instead make them prefer some more specific instructions about ow to implement CI.

**Increased pressure.** Pressure from management as well as a feeling of lacking the experience to adapt to new demands can overwhelm a team. Therefore, it would be in the best interest to implement gradual changes rather than trying to do everything at the same time.

# *Bibliographie*

[1] Grady Booch. *Object Oriented Design with Applications.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.

[2] John Ferguson Smart. *Jenkins : The Definitive Guide.* O'Reilly Media, Inc., 2011.

[3] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration : A case study. In Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen, editors, *Product-Focused Software Process Improvement*, pages 17–32, Cham, 2014. Springer International Publishing.

## Image Sources

1. jenkins : me
2. travis : http ://mahugh.com/2016/09/02/travis-ci-for-test-automation/
3. buildbot : http ://docs.buildbot.net/0.8.3/System-Architecture.html
4. buildbot-worker-workflow : http ://docs.buildbot.net/0.8.3/BuildSlave-Connections.html#BuildSlave-Connections
5. nextdoor : https ://engblog.nextdoor.com/3-hard-lessons-from-scaling-continuous-deployment-to-a-monolith-with-70-engineers-99fb6dfe3c38