The Unix-like Build Pattern

Bruno P. Kinoshita[1]**, Eduardo Guerra**[2]

[1] TupiLabs
Sao Paulo – SP – Brazil

[2] Instituto Nacional de Pesquisas Espaciais (INPE)
Sao Jose dos Campos – SP – Brazil

bruno@tupilabs.com, guerraem@gmail.com

Abstract. *The time that software takes to get built is a challenge for many development teams. Different ways to reduce this time have already been used in software projects, such as compiling permutations and executing unit tests in parallel. The pattern described in this paper, called Unix-like Build Pattern, can be used to reduce the build total execution time by splitting it into smaller parts.*

Resumo. *O tempo que um software leva para ser construído é um desafio para muitos times de desenvolvedores. Diferentes maneiras de diminuir este tempo já foram aplicadas em projetos de software, como permutações na compilação do código-fonte e execução de testes unitários em paralelo.O padrão apresentado neste artigo. chamado de Unix-like Build Pattern, pode ser utilizado para reduzir o tempo total de execução do build quebrando-o em partes menores.*

Introduction

Building software is an important part of software development. There are several ways of doing it but a series of good practices and tools can be used to help you to achieve better results. The pattern described in this paper can reduce the build execution time, specially when the build is comprised of several tasks (aka build steps).

This paper was written to describe an existing pattern, already in use in many organizations and open source projects. It can be used by developers, devops, build managers and systems engineers to reduce the build execution time.

This paper focus on Jenkins build server and Java dependency and life cycle build tools. However, this build pattern can be considered independent of build server, programming language and build tools. In the next sections we introduce Continuous Integration, Jenkins, the Unix way and the build pattern using the Alexandrian pattern format [Alexander *et al* 1977].

Continuous Integration and builds

Continuous Integration (CI) has been part of software development for many years, but has gained more attention with agile techniques and cloud computing. Martin Fowler's definition is probably one of the most cited, and defines CI as "(...) a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily" [Fowler 2013].

[Smart 2013] and [Titus, Canino-Koning 2013] also define CI as a system that build and test software automatically and regularly (e.g.: with a simple scheduled build), adding more value to the software, bringing advantages to the software development teams and to the end-users. Software development teams have already understood the importance of CI. Tools like [Atlassian Bamboo 2013], [Apache Continuum 2013], [CruiseControl 2013], [Apache Gump 2013], [Hudson CI 2013] and [Jenkins CI 2013] are essential for most of these software development teams.

The set of good practices described by [Fowler 2013] includes:

1. Maintain a single source repository
2. Automate the build
3. Make your build self-testing
4. Everyone commits to the mainline every day
5. Every commit should build the mainline on a integrate machine
6. **Keep the build fast**
7. Test in a clone of the production environment
8. Make it easy for anyone to get the latest executable
9. Everyone can see what's happening
10. Automate deployment

Note that the sixth item – "keep the build fast" – is directly related to the main goal of the Unix-like Build Pattern.

In a big project the build time can consume a significant amount of time. And the addition of new steps to the build, like executing tests on multiple environments, source code analysis and generating new reports can also contribute to making the build slower.

Taking into consideration size, complexity and steps required to build software; a build server is a very important part of Continuous Integration [Fowler 2013]. The build server acts as a monitor to the source repository [Fowler 2013]. It reacts to source code commits and triggers the necessary steps to build the software. All these build steps can take a long time to execute.

The Unix-like Build Pattern

Also known as **Parallel Builds or Parallel Build Steps Execution**.

It is not hard to find real projects having separate but related jobs [Smart 2013] for building the software. A common case are products that are a combination of several modules or components [Thiim, Hvatum 2012]. Besides having several parts, a software can also require the execution of several steps to get built.

These build steps can vary according to programming language, framework, components or API's used. Assessing the project structure (e.g. static code analysis), splitting the source code compilation into multiple parts (e.g. GWT compiling permutation), running tests in parallel (e.g. JUnit 4.7+) and sending notifications (e-mails, SMS's, network alerts, etc) are examples of common build steps.

*   *   *

How to optimize a build execution that is composed by several tasks?

Unix developers and users have axioms that define how tools and scripts are created and used together. This principle is called 'the Unix way', or 'the Unix philosophy'. It is based on the principle of modularity, with each program doing one thing well, and connecting programs using the output of one program as the input to another.

This build pattern applies the Unix way on the software structure. It can be used to build software with a long execution time and/or allow more builds to be executed simultaneously. In Figure 1 we can see the general idea of this pattern, and a comparison between a monolithic and a Unix-like build.

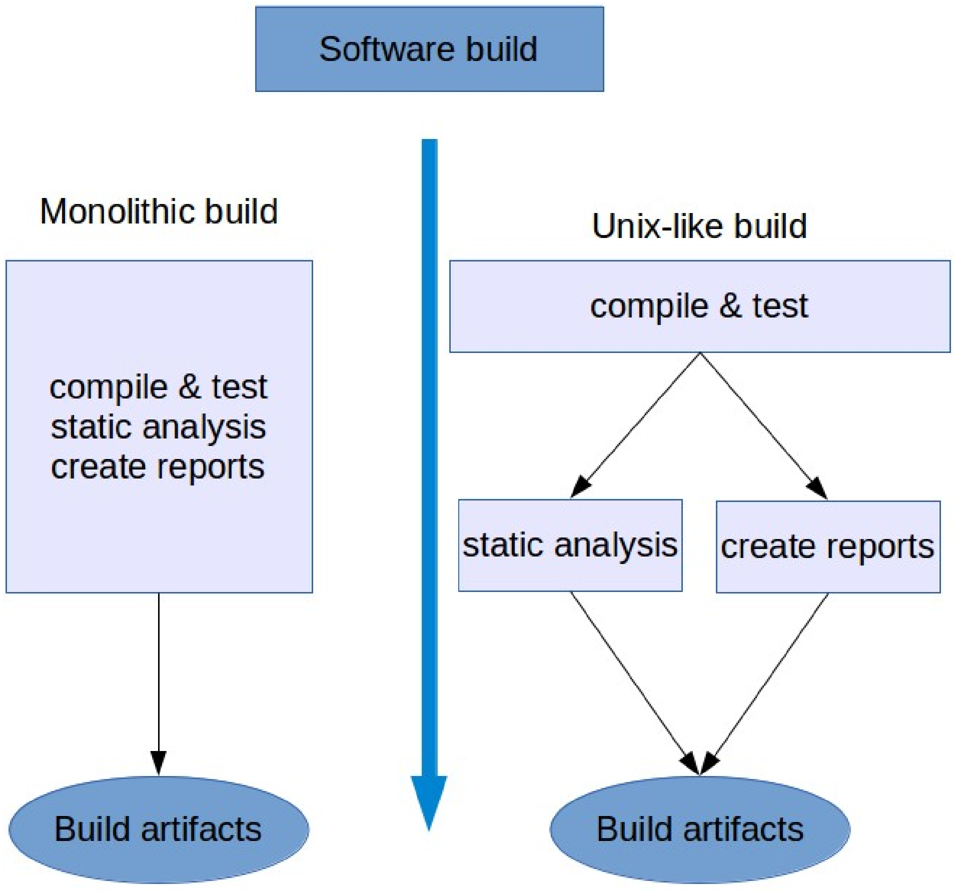


Figure 1. Unix-like build pattern

By applying this pattern to your builds you can reduce the build total execution time by allowing the scheduler to run build steps simultaneously on multiple CPU cores or computers, reducing the turnaround time.

As you can see, the build is split into smaller parts, similar to the Unix way. The output of build steps is used as the input of another build step. And the parts of the build can be easily replaced.

Therefore:

* * *

Break the build into smaller parts, executing in parallel independent tasks.

Jenkins (previously known as Hudson is "an open source Continuous Integration tool written in Java". Even though it is written in Java, it is programming language and runtime agnostic, being able to build software written in different programming languages.

Looking at Figure 2 you can understand how the build queue wait time can be reduced. While the build server is running the Build #11, it can start running the tests for Build #10 job in parallel. Thus reducing the time that builds have wait in the queue for an available execution slot.

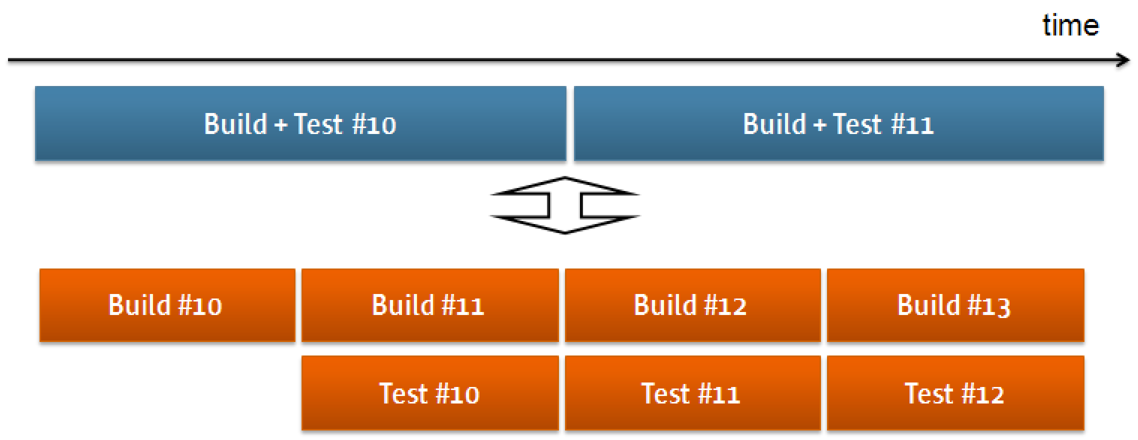


Figure 2. Running multiple builds [Jenkins CI Wiki 2013]

[Titus, Canino-Konning 2013] separate build servers in those that have master and slaves, being able to run builds in parallel, and those that have only a master, centralizing the whole work. Using this classification, Jenkins fits to the "*hybrid mode*" category. In other words, it can be used as a central build server or running jobs distributedly.

Using plug-ins, and/or building a master/slave Jenkins instance, you can schedule jobs to run in parallel in one or multiple computer. This is an important and useful feature when you have to break your build into smaller parts, since you can decide where you would like to execute your builds.

A build job (or simply job) in Jenkins can be either a single step/task or can do a number of things, from compiling and testing code, to running special reporting tools or integrating with other systems. The vanilla version of Jenkins is able to handle relation upstream/downstream jobs. It means that you can define one job to trigger a subsequent one, and so forth, until you have successfully built your software. There are also plugins that can further extend this functionality by creating build pipelines or by coordinating multiple jobs executions.

* * *

Relationship to other patterns.

The Component Baselines build pattern, described by [Thiim, Hvatum 2012], can be used to break the application into smaller parts, the component baselines. Since it reduces the size of the project that needs to get build, you will be able to build only the isolated components baseline, instead of the whole project.

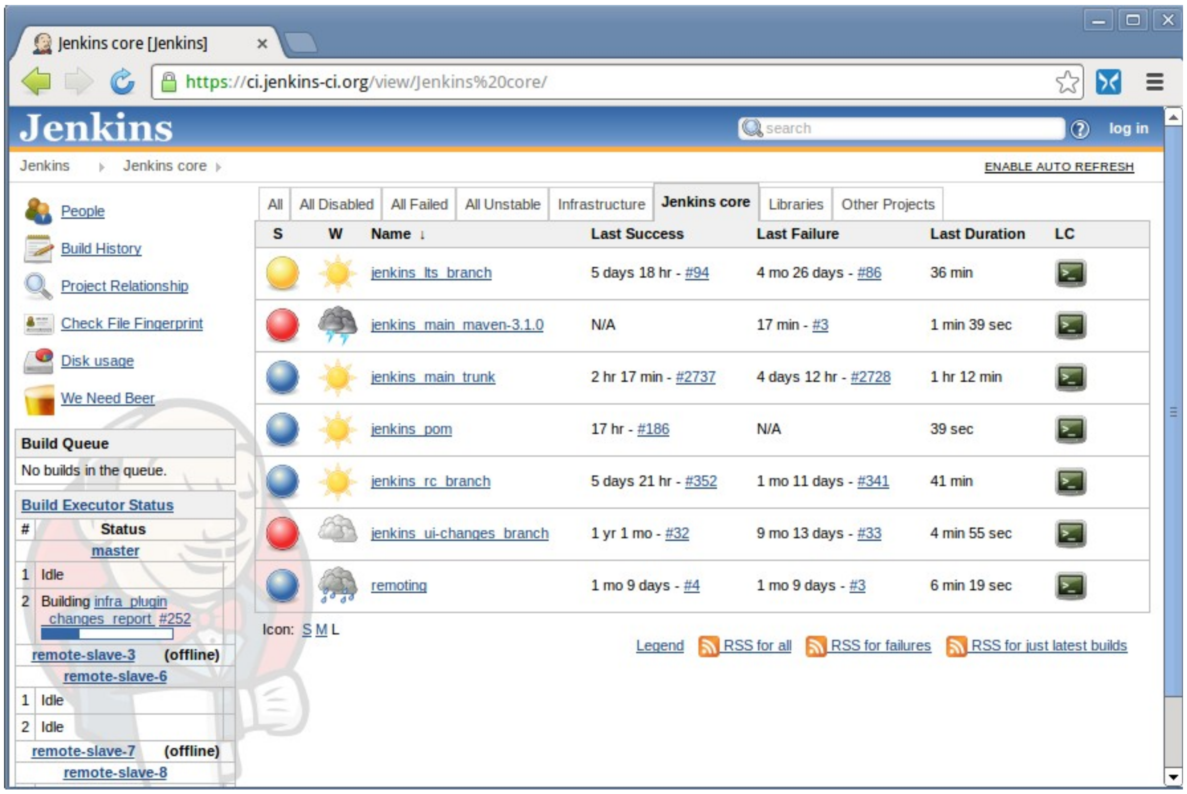
Another pattern that can be used to find a suitable suite to apply the Unix-like build pattern is the Sparse Workarea pattern [Thiim, Hvatum 2012]. With the sparse work area, you can work only on the code being modified, reducing what needs to be build too, thus reducing your total execution time during development.

Breaking your project into smaller parts and scheduling some of these parts to build in specific time slots, is exactly the same that is done in the Time Slicing build pattern [Simpson 2011]. However, while the focus of the Unix-like build is modularity and time saving - as in the Unix way - the Time Slicing pattern is focused in efficient resource utilization.

* * *

Unix-like Build Pattern in real life.

Jenkins CI (the Open Source project) has a Jenkins instance that builds the Jenkins Core. There are always at least two jobs in there, one for building the main branch, and another for the LTS (Long Term Supported) branch. But there are other jobs used for branches that are created for specific features (e.g. changes in the UI).


Jenkins core [Jenkins]

https://ci.jenkins-ci.org/view/Jenkins%20core/

Jenkins › Jenkins core ›

ENABLE AUTO REFRESH

People
Build History
Project Relationship
Check File Fingerprint
Disk usage
We Need Beer

Build Queue
No builds in the queue.

Build Executor Status

#	Status
	master
1	Idle
2	Building infra_plugin changes_report #252
	remote-slave-3 (offline)
	remote-slave-6
1	Idle
2	Idle
	remote-slave-7 (offline)
	remote-slave-8

All | All Disabled | All Failed | All Unstable | Infrastructure | Jenkins core | Libraries | Other Projects

S	W	Name ↓	Last Success	Last Failure	Last Duration	LC
●	☀	jenkins_lts_branch	5 days 18 hr - #94	4 mo 26 days - #86	36 min	
●	⛈	jenkins_main_maven-3.1.0	N/A	17 min - #3	1 min 39 sec	
●	☀	jenkins_main_trunk	2 hr 17 min - #2737	4 days 12 hr - #2728	1 hr 12 min	
●	☀	jenkins_pom	17 hr - #186	N/A	39 sec	
●	☀	jenkins_rc_branch	5 days 21 hr - #352	1 mo 11 days - #341	41 min	
●	☁	jenkins_ui-changes_branch	1 yr 1 mo - #32	9 mo 13 days - #33	4 min 55 sec	
●	🌧	remoting	1 mo 9 days - #4	1 mo 9 days - #3	6 min 19 sec	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Figure 3. Jenkins core builds [Jenkins CI 2013]

Another example is creating a separate job for SonarQube code analysis. SonarQube is a quality assurance dashboard tool, that collects several static and dynamic code metrics, such as cyclomatic complexity, lines of code, test coverage among others. While SonarQube is a very useful tool, it can take a while, since it may have to index each source code file several times to collect and present metrics.

A common mistake is to configure a job in Jenkins that executes SonarQube for each new build, occupying a slot in the build queue for longer than necessary. Applying the unix-like build pattern, one could save time by scheduling the SonarQube execution to only certain periods of time, such as during the nightly build, or before a new release is cut.

Projects hosted at Apache can ask for a Jenkins build at http://builds.apache.org, and a SonarQube analysis project at http://analysis.apache.org. Jenkins builds at Apache are, in general, responsible for compiling the code and running tests, and are triggered for every commit. The SonarQube projects are built in another Jenkins instance on a daily basis only if the project changed since last run.

Conclusion

In a competitive environment, with other rival teams and/or corporations, the time to build the project can be a key factor. It can also be demotivating for software developers that have to wait for a long time for the build to finish.

The pattern described in this paper is a well-known pattern among software developers, devops, system engineers and other professionals. We called this pattern the Unix-like Build Pattern since it follows the Unix way – or Unix philosophy – by splitting the build into smaller parts rather than having a single monolithic build.

These smaller parts can be build steps or other jobs. This pattern allows you to execute these build steps in parallel, taking advantage of multiple CPU core architectures, as well as Jenkins installations with a master/slaves architecture.

Finally, another benefit is giving chance to other jobs to be executed by the build tool while a task is being performed on another CPU. This can reduce the time that builds have to wait in a build queue to be executed, increasing the build server throughput.

References

Alexander, C., Ishikawa, S., Silverstein, M. (1977) "A Pattern language: towns, buildings, construction", Oxford, Oxford University Press.

Apache Ant (2013), http://ant.apache.org, Jul

Apache Continuum (2013), http://continuum.apache.org/, Jul

Apache Gump (2013), http://gump.apache.org/, Jul

Apache Ivy (2013), http://ant.apache.org/ivy, Jul

Apache Maven (2013), http://maven.apache.org, Jul

Apache Wiki (2013) "Sonar Instance at Apache" http://wiki.apache.org/general/SonarInstance. Jul

Atlassian Bamboo (2013), https://www.atlassian.com/software/bamboo/, Jul

Composer (2013), http://getcomposer.org/, Jul

CruiseControl (2013), http://cruisecontrol.sourceforge.net/, Jul

Fowler, Martin (2013) "Continuous Integration", http://www.martinfowler.com/articles/continuousIntegration.html, Jul

GNU Make (2013), http://www.gnu.org/software/make/, Jul

Hudson (software) (2013), "Hudson-Jenkins split", http://en.wikipedia.org/wiki/Hudson_(software)#Hudson.E2.80.93Jenkins_Split, Jul

Hudson CI (2013), http://hudson-ci.org/, Jul

Jenkins CI Wiki (2013) "Splitting a big job into smaller jobs", https://wiki.jenkins-ci.org/display/JENKINS/Splitting+a+big+job+into+smaller+jobs. Jul

Jenkins CI (2013), http://jenkins-ci.org/, Jul

Jenkins Clone Workspace SCM Plug-in (2013), https://wiki.jenkins-ci.org/display/JENKINS/Clone+Workspace+SCM+Plugin, Jul

Jenkins Copy Artifact Plug-in (2013), https://wiki.jenkins-ci.org/display/JENKINS/Copy+Artifact+Plugin, Jul

Julian, Simpson (2011), "Build Patterns to Boost your Continuous Integration". Methods & Tools, Spring 2011.

McIlroy, M.D., Pinson, E. N., Tague, B.A. (1978) "UNIX Time-Sharing System: Forward". Bell System Technical Journal.

Phing (2013), http://www.phing.info/, Jul

Raymond, Eric Steven (2003), "The Art of UNIX Programming (The Addison-Wesley Professional Computing Series)". Addison-Wesley.

Smart, John Ferguson (2011) "Jenkins: The Definitive Guide". O'Reilly Media.

SonarQube (2013), http://www.sonarqube.org/, Jul

Thiim, Ralph, Hvatum, Lise (2012). "Organizing and Building Software - Patterns for effective management of large and complex code bases". Pattern Languages of Programs Conference (PLoP) 2012.

Titus, C., Canino-Koning, Rosangela (2012) Continuous Integration, "The Architecture of Open Source Applications – Elegance, Evolution, and a Few Fearless Hacks". lulu.com.