# Maven and its handling of build dependencies

Aristotelis, Kotsias
kotsias@kth.se

Sandro, Lockwall Rhodin
sandror@kth.se

April 2020

**Abstract**

Build tools are an essential part of the DevOps software development lifecycle. The most simple build tools are just a bash script or a shell script. These scripts consist of a series of terminal commands or functions that can be executed from the command line. However, updating files that have changed or executing them are tricky with just a shell and this is where automated build tools come in and add file management and more. Popular Java build tools are Ant, Gradle and Maven. This report, attempts to present the importance of build tools in DevOps and also examine the Maven build tool and its handling of build dependencies.

# 1 Introduction to build tools

Build tools automate the "translation" of the source code of a software program into executable or usable form. Specifically, build tool is a catch-all term that refers to the chain of processes that take place in order to get a piece of software set up and ready for use. There are three use cases where build tools are essential; production deployment, continuous integration (CI) and developer environments [1].

Production deployment refers to the process where the codebase is set up once and no changes are made after. The role of build tools in production deployment is simple as they are used to generate executable environment by checking out and compiling the source code. During this process, the build tool is responsible for running arbitrary commands and using the results of one command in another. In CI, tests are running together with building, every time there is a new commit in a repository. The difference from production deployment is the frequency of building, as CI builds happen more frequently. Therefore, build tools in CI need to be faster. In developer environments, the build tool should be able to re-compile the source code and re-generate any file and therefore updating the system so as the developer is able to interact with the system after the changes he make. [1]

The processes that the build tools automate are; configuration of the build environment, synchronization of source code with the deliverables by execution of the necessary build commands and running the test suites in order to ensure the correct behavior of the deliverable system [2]. During the build phase, the build tool will determine which commands need to be executed according to the build specifications. The advantage of build tools against the naive old-school build scripts, is that the build tools support incremental builds. Specifically, this enables the build tools to optimize the build by computing and executing the minimal number of commands based on the changes that incurred since the last build [3]. For instance, if a developer slightly modifies the source code of a system, i.e. adds an if-else statement, then rebuilding the system according to incremental build is faster than a clean build. This is because clean build would require to delete everything from previous builds and start from scratch while incremental build saves the information from previous builds and re-uses it every time a new build is triggered. In figure 1, famous build tools are presented.

# 2 Relation to DevOps

DevOps is about the practice of technologies and tools, rather than a technology, in order to deliver high software quality. DevOps revolves around the concepts of continues integration, automation, continuous delivery, microservices and collaboration [4]. However, non of these are possible without the use of the right software tools. As it is shown in Figure 2, the DevOps tool chain consists of

| Other DevOps build tools | | | |
|---|---|---|---|
| Apache Ant | Gulp | NAnt | Travis CI |
| Broccoli | Hudson | Packer | UrbanCode Build |
| Buildbot | Jam | QuickBuild | Visual Build |
| BuildMaster | LuntBuild | Rake | Visual Studio |
| FinalBuilder | Make | sbt | |
| Buildr | Meister | SSH | |
| CMake | Microsoft Build Engine (MSBuild) | TeamCity | |

Figure 1: Build tools
source by: https://www.smartsheet.com/devops-tools

code, build, test, release, deploy, operate, monitor and planning and for each step different software tools are used. No single tool can do the work for all of the software development stages. Therefore, build tools are essential part of the DevOps tool chain and their role is fundamental. There are many different build tools, depending on the programming language that is used on the project, but all of them share a common goal which is to process and build the source code into the desired executable format and to automate the tasks of compiling testing and deployment.

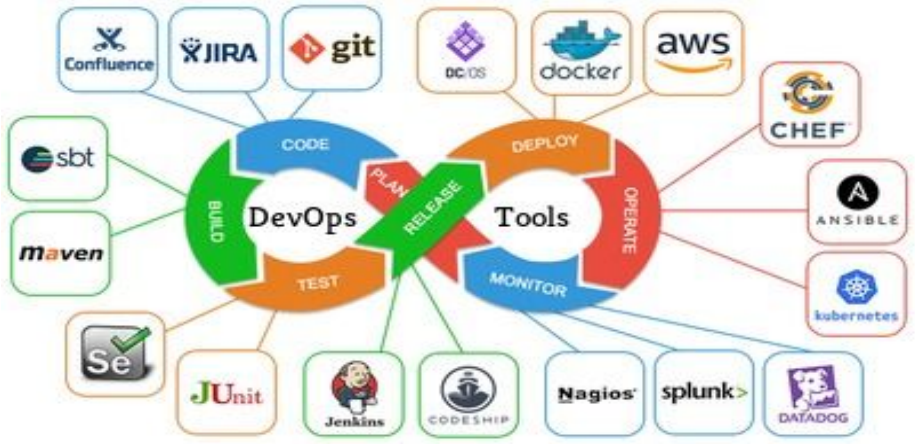


Figure 2: DevOps Tool Chain
source by: https://gr.pinterest.com/pin/802696333569666729/?autologin=true

# 3 Definition of Build dependencies

All build tools require some way of dealing with a projects that rely on other software to function, these types of software are known as dependencies. They are defined as being essential software on which a project depends to either compile or run. A project could be built without any major, if any, dependencies at all, but for larger projects this is not often the case. In turn, when a larger project is built, dependencies must be checked for, and collected if not already available.[5]

# 4 Structure of the Maven Build dependencies

The Maven dependencies are setup in a POM file, one that is created when first setting up a Maven project. The POM file functions as an extensive guide when the Maven project is built to detail how it should be built, and for example if specified, download all necessary dependencies if they are not already available. The standard, generated POM file is filled with a great amount of information such as where the project is supposed to be built, but for the purpose of this essay only the dependency section will be treated [6].

The structure of the dependency management in the POM file can be split into a number of parts. First, any and all dependencies are stated within a <dependencies>-tag and a </dependencies>-tag. While any individual dependency are similarly within <dependency>-tags. This is important for the structure of the POM file itself, but not necessarily the most useful information [7][8].

The more interesting information is found under the <dependency>-tags, since there are a number of other tags contained within it. In particular, the <groupId>, <artifactId>, <version>, <type>, and <scope> tags [7].

The <groupId> tag is a unique project identifier. One that should follow all of the Java package naming conventions. For dependency work, it is used to identify a specific dependency that needs to be gathered from a specific project [9].

The <artifactId> tag generally identifies what kind of specific jar file should be collected from the specific project identified with the <groupId> tag. In that sense it defines what actually should be collected from the project [9].

<version> then specifies which version of the jar file should be collected. Since many projects carry many different versions of their builds as their projects develop over time [10].

However, sometimes what you're collecting isn't a jar file. In that case you need to specify what kind of file you're trying to collect. This is done through the <type>-tag. Generally these are archives, but plugins can be used to define new types to be used [10].

The <scope> tag indicates when the dependency should be used. Sometimes things only need to be used when compiling the software, other times only when running or testing the software [7].

A standard dependency can be defined only using the <groupId> <artifactId> and <version> tags, as this is potentially enough information to identify the dependency. [6][7]

# 5 Dependencies and Repositories

Once the dependencies are setup, utilising at least the minimal number of tags, and the Maven project is built, one might ask oneself: How do all of the dependencies get resolved? Well, there are two ways for dependencies for Maven to be resolved, both of which are setup through the POM file. Dependencies that have local files and dependencies that can be found on a remote repository. Local dependencies are rather simple, by default when creating the Maven project, a directory for the dependencies is created and any dependency put into that folder can be grabbed and used by the project. In the beginning however, there are no dependencies stored in the local directory so they have to be obtained somehow [11].

Obtaining dependencies from non-local sources is primarily done through remote repositories. They can be setup utilising the <repositories> and <repository> tags respectively. Each remote repository needs at least an <id> tag and a <url> tag, providing an identification name and a link to the repository [11].

In the default POM file, a repository is already setup to handle the dependencies you have for your Maven project [6]. This is the Maven Central repository and it is commonly used by more than just Maven projects. It is for example one of the recommended repositories for build tools similar to Maven such as Gradle [12]. The repository works as a storage of some of the most commonly used dependencies, with over 4 million indexed jar files at the time of writing [13][14].

Adding Maven dependencies is often made easier with access to the Central Maven Repository, using the officially recommended search engines (repository.apache.org, search.maven.org and mvnrepository.com) [15]. If you navigate to any of these search engines using a browser you can find and collect necessary jar files and quite likely also find the dependency string needed to be added under the <dependencies> tag to have Maven collect the dependency automatically when building.

# 6 Why to use Maven

The main benefit of using Maven is the dependency management that offers during the development of Java applications. For small and easy projects, dependencies configuration is not hard and rarely anyone has to do anything with it. However, when a larger and more complicated project is developed, managing the dependencies manually by hand is rather trivial without the use of a build tool such as Maven. Specifically, Maven offers tools that deal with dependency management and offers the opportunity to the user to control the version used in transitive dependencies, analyze the dependency tree and control the converge across modules. Moreover, due to Maven's central repository, all developers use the same jar dependencies and also the source distributions' size is reduced due to the fact that the needed jar dependencies can be pulled from the central repository.

Furthermore, Maven deals with transitive dependencies meaning that when an external dependency is included in a project, the developer does not need to download all the other dependencies that the external dependency is dependent on [7]. This is because Maven will automatically download all the necessary jar files that are needed and also takes care of downloading the right versions. This is a powerful feature of Maven as it is time efficient and build errors that could occur due to incorrect versions of dependent libraries are eliminated.

Compared to other build tools such as Ant, Maven provides the user with preconfigurations, minimizing the amount of time a user has to spend setting up the build tool. Not only that, Ant simply does not utilise repositories. It requires that all dependencies to be used have to be in a user set folder from where they will be used when configured to be used appropriately during the building. This means, that compared to Maven which will download dependencies as necessary, the user will have to manually download every dependency to utilise them in Ant.[16][17].

Additionally, Ant follows a task based approach meaning that the user has to specify to Ant exactly what to do and when to do it. Specifically, Ant's build files contain explicit instructions that tell Ant where to find the source code and compile it, then copy it and eventually compress it. On the other hand, Maven follows a declarative approach, meaning that the user all he has to do is to use the auto-generated pom.xml file and put the source code in the default directory, as Maven will take care of the rest [18].

# 7 Discussion

As it has already been described, Maven is central repository for dependencies and a powerful management tool that automates project's build which is based on POM. The POM file contains information about the project configuration

such as source directory and dependencies. This section will reflect upon the benefits and potential drawbacks of Maven and its dependency handling.

The dependency handling is rather simple, with not too many tags to worry about when constructing the dependency. An added benefit of utilising a Maven project with the Maven Central Repository is that the Maven Central repository can provide dependency information to be added under the <dependencies> tag. This could reduce the likelihood of a user inputting the wrong information and therefore failing to gather the dependency.

There aren't necessarily any real drawbacks with Maven and its handling of dependencies, other than perhaps that it could be easy for a user to fail to access to correct dependency if they are manually inserting the needed dependency information. This lack of drawbacks to Maven's handling of dependencies becomes especially clear when compared to the likes of Ant which forces you to do even more things manually, potentially making it even easier for the end user to properly setup their project.

# 8 Conclusion

In this report, Maven's handling of dependencies has been examined and a brief comparison between Maven and Ant has peen presented. Maven offers a basic POM file setup allowing for easy additions of dependencies, if needed. Together with easy to understand tags for setting up custom dependencies and with Maven also recommending search engines to find the desired dependencies, that users might want to add to projects, showing this desire for an easy to use service. In conclusion, Maven shouldn't be thought strictly as a build tool but as a highly configurable project management tool which uses a well-defined project object model to describe the project.

# References

[1] L. Haoyi. (2020, Apr.) What's in a build tool? @ONLINE. http://www.lihaoyi.com/post/WhatsinaBuildTool.html.

[2] C. Desarmeaux, A. Pecatikov, and S. McIntosh, "The dispersion of build maintenance activity across maven lifecycle phases," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 492–495.

[3] A. Ghazarian. (2020, Apr.) Incremental and fast builds using ant @ONLINE. https://www.javaworld.com/article/2072134/incremental-and-fast-builds-using-ant.html.

[4] (2020, Apr.) What is DevOps? - Amazon Web Services (AWS) @ONLINE. https://aws.amazon.com/devops/what-is-devops/. Library Catalog: aws.amazon.com.

[5] D. Dilshan. Do you know maven ? a dependency manager or a build tool ? what is pom ? https://medium.com/@dulajdilshan/do-you-know-maven-a-dependency-manager-or-a-build-tool-what-is-pom-bd7dd8b43e80. Last Accessed: 2020-04-25.

[6] Introduction to the pom. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html. Last Accessed: 2020-04-09.

[7] Introduction to the dependency mechanism. http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html. Last Accessed: 2020-04-07.

[8] Maven – dependency management. https://howtodoinjava.com/maven/maven-dependency-management/. Last Accessed: 2020-04-09.

[9] Guide to naming conventions on groupid, artifactid, and version. http://maven.apache.org/guides/mini/guide-naming-conventions.html. Last Accessed: 2020-04-07.

[10] Maven. https://maven.apache.org/ref/3.6.1/maven-model/maven.htmldependency. Last Accessed: 2020-04-07.

[11] Introduction to repositories. https://maven.apache.org/guides/introduction/introduction-to-repositories.html. Last Accessed: 2020-04-17.

[12] Declaring repositories. https://docs.gradle.org/current/userguide/declaring_repositories.html. Last Accessed: 2020-04-17.

[13] Central repository. https://mvnrepository.com/repos/central. Last Accessed: 2020-04-21.

[14] Maven - repositories. https://www.tutorialspoint.com/maven/maven_repositories.htm. Last Accessed: 2020-04-21.

[15] Frequently asked technical questions. http://maven.apache.org/general.html. Last Accessed: 2020-04-21.

[16] F. Hernandez. Difference between ant and maven. https://examples.javacodegeeks.com/enterprise-java/maven/difference-ant-maven/. Last Accessed: 2020-04-28.

[17] baeldung. Ant vs maven vs gradle. https://www.baeldung.com/ant-maven-gradle. Last Accessed: 2020-04-28.

[18] T. O'Brien, *Maven: The Definitive Guide*. O'Reilly Media, 2008.