OpenMake Software

Why Use Meister and Not Ant or Maven

A whitepaper review of the differences between Meister and Ant/Maven

By Tracy Ragan, COO, OpenMake Software

Contents

Summary

Building Java code involves more than pointing and clicking your way through an IDE such as Eclipse. Continuous Integration requires that you build outside the IDE and have the ability to hand-off code to a different team for compiling, packaging and deployment – what is often referred to as DevOps. This paper explores the benefits of using Meister as compared to open source solutions such as Ant and Maven for managing the creation of Java binaries.

The Challenge

Organizations are analyzing ways to improve the process by which applications are developed. They are looking at the compile, package and deploy steps of the software build and searching for ways to be more efficient. Historically, developers working on the distributed platforms have relied on hard-coded, static build and deploy scripts, thusly putting the full responsibility of creating and deploying binaries on the shoulders of developers. As distributed applications, such as Java, have become more mission critical, there is a need for standardization. Moving the application through the lifecycle with shared knowledge and consistency is the path to standardization.

"Historically, developers working on the distributed platforms have relied on hard-coded, static build and deploy scripts, thusly putting the full responsibility of creating and deploying binaries on the shoulders of developers."

Deciding on Scripting

For the last 30 years, distributed platform teams have been coding scripts to execute software compiles. Developers commonly face the challenge of creating a "build" late in the project schedule and they reach for what is quickly available. No"Request for Proposal" is developed, requirements are not documented, no "Proof of Concept" is done, budgeting is avoided, and best yet, there are no sales reps to be contacted. For Java, a developer simply downloads the Ant scripting language and goes to work. As things get more complicated, they might use Maven and add Jenkins to help the process. In other words, no real decision making is done. The build step is put together in an ad-hoc method often with mixed results.

This process doesn't allow for a corporate level assessment of what functionality is actually needed from the build process.

"The challenge for developers who advocate for a build and deploy process that is built upon static scripts is to understand the challenges of the entire organization. "

Organizations need the ability for someone other than the script author to support the process and generate the reports needed for audits. Critical requirements of the development team, such as the ability to accelerate the compile steps, manage dependencies or build incrementally - are not reviewed.

Ultimately this ad-hoc process becomes a full-time job for a key developer to support and maintain, and the process does not address the needs of the entire organization, it only addresses what the development team needs to continue development.

The Bigger DevOps Picture

Relying on tools such as Ant/Maven and Jenkins to get the Development CI Build and Deploy done may be sufficient for the development teams. The challenge for developers who advocate for a build and deploy process that is built upon static scripts is to understand the challenges of the entire organization. Building across a lifecycle with different technology stacks, guaranteeing matching source to production binaries, audit reports, securing control over production servers, delivering an emergency fix without introducing new features, these are all requirements that go beyond what developers design in their scripted processes. Streamlining the Build to Deploy process must involve the entire organizations requirements, and this is precisely what DevOps promises to deliver. Automation is key and cannot rely on a static script controlled and maintained by development.

OpenMake Meister vs. Ant and Maven Comparison

OpenMake Meister compares more closely with Maven than it does with Ant. Ant is simply a build scripting language that allows the developer to code the build intelligence that is needed to build and deploy software. The power in Ant is that it is highly flexible and handles uniqueness. The purpose of Maven is to reduce the number of Ant Scripts down to a set of standard scripts, or project object models (POM), and plug-ins that are shared across teams for building java binaries. The power in Maven is that it "shields the details" of the build from the user thus simplifying the effort of creating Ant Scripts. The problem with Maven is that it does not often support the uniqueness required for each team. For this reason, you can add Ant scripting to your Maven objects. When this begins to occur, Maven's standardization is compromised and the Maven process is not an Ant and Maven process.

Regardless if you are using Ant or Maven, the fact remains that the intelligence built into the scripts come from one of the developers. Meister does not rely on the intelligence of a developer and delivers both the flexibility of Ant along with the standardization of Maven. In addition, it delivers insightful reporting, dependency management that is derived (not coded), acceleration and parallelization. Below is a comparison chart that reviews a small set of features important in a Build Management process. There are many other features to consider, but when comparing OpenMake Meister to Ant and Maven, these variables must be understood.

Features	Meister	Ant and Maven
Integration with Eclipse – building inside and outside of the Eclipse IDE.	YES Meister supports an Eclipse Plug-in that generates build.xml for a single Jar file or a build.xml file that supports the compiling of an application comprised of multiple Jars, Wars and Ears. Meister utilizes Ant to execute the compile process, but does not require the developer to code the Ant build.xml. Editors and syntax checking are not needed. Meister is more efficient then native Ant as it removes the requirement to code a project build.xml file and a complete application build.xml file.	Ant- YES Eclipse includes a feature called "code assist" that helps you write your Ant script, including the checking of syntax. In essence it does its best to help you "generate" the Ant build.xml file that is needed to build the Eclipse Project outside of the IDE. Maven- YES: Maven has integrations to Eclipse (under different names). The integration allows you to create your new Maven Projects based on Archetypes, execute a maven build and other tasks that are part of the Maven Command Line.
Enforces standardization and facilitates repeatability	YES Because Meister generates the build.xml file, the standards around how the compile/archive steps are implemented can be loosely or tightly enforced. Compile/Archive options, the use of external or internal repositories, the management of the technology stack across the lifecycle and access privileges can be defined across the enterprise. Controlling the method of building Jars, Wars and Ears across the enterprise creates a more repeatable process from the local pre-flight build, CI build, QA build and pre-production build.	Ant-NO Because ANT is scripted using XML, each developer can modify build behavior and flags as they see fit. From development team to development team the build process can vary greatly because there is no centralized knowledge base for storing standards. Maven- Partially Maven attempts to solve this problem through the use of "archetypes" and POM files. While this initially may offer some standards, the standards can easily be moved away from as each team customizes their Maven process. In addition, because this is a scripted process, it is difficult to enforce the standards. Changes in the Maven POM, or even the addition of Ant scripting to address a certain requirement breaks the standardization.

Automatic CI incremental builds and deploys	YES Meister performs automatic dependency analysis through code scanning to make intelligent build decisions. This means that if an object is already up to date, it is not re-built for example a .java to .class. In addition, Meister automatically determines java source to Java Jar, War or Ear dependencies which means that if a Jar, War or Ear file has been updated for any reason, Meister can re-build the entire application based on only the changes. In essence Meister can perform incremental build management on the entire Java application structure automatically without a person's input.	Ant-Partial Ant can perform .java to .class incremental builds; however in order to achieve dependencies between a java source and a java jar, the use of the "Depend" tag is required. This is done manually and is not automatically determined by Ant. Because the Depend tag is manually coded, it can easily become stale as the application changes and the hard-coded dependencies are not maintained. For this reason, the Depend statement is not often included in Ant scripts as there is a high risk in relying on the Depend tag and having a build fail as the incremental processing is incomplete. Maven-Partial Maven does not support incremental builds and the recommendation is to do a clean build each time. It can help with managing transitive dependencies and you can add Ant scripting using the "depend" tag. The core issue is that neither Ant or Maven does low level code scanning and these lower level dependencies are hidden from Ant and Maven. A person can sort them out and code for this, but this level of understanding can be difficult for even the smartest developer. The use of a computer program for scanning code and dependency relationships is what is required.
Support for builds across the lifecycle, Agile methodologies and parallel development environments.	YES Meister generates the appropriate build.xml file for each stage of the lifecycle or parallel stream. There is no need to have different build.xml files managed for multiple stages or streams. The generated build.xml file can be versioned for auditing.	Ant- NO Because Ant is static and has no concept of a lifecycle, an Ant script is written and maintained for one version of the application at a time. As the application changes, the Ant build.xml file must be updated for that version and managed with the code resulting in multiple versions of the build.xml. weiter nächste Seite ...

		Maven-No Even though Maven supports a lifecycle using “goals”, what the lifecycle does is different from supporting a build for test that has different components than production. What Maven’s lifecycle delivers is more of a workflow supporting phases and goals that can be included as part of your Maven build, from running code validations, compiles and deploys. If the logic inside the compile is different for Test and Production, different Maven scripts are managed and executed at the compile step. Maven’s lifecycle can be compared to products like Jenkins or OpenMake Software’s Mojo.
Accelerated Java Builds	YES Because Meister performs dependency management on the entire Java application structure, it can accelerate builds using parallelization. Meister can be told to perform a Java build calling the compiler simultaneously. Meister customers have reported running a build with 300 java compiler instances and have been limited by only machine resources.	No for both Ant or Maven has no parallelization capabilities.
Management of low-level or transient dependencies	YES Meister can reference external repositories to automatically discover what your own dependencies may require. Meister can also enforce the use of specific versions of these dependencies that can be carefully controlled through the Meister Dependency Directory. In addition, an organization may have their own set of “transitive” dependencies referenced in reusable components. Meister can manage these types of custom transient dependencies as well.	YES for both Ant tasks and Maven Plug-ins can manage a Transitive Dependency model for automatically discovering what your own dependencies rely on. You can also create a private repository that can include the versions of these dependencies that your organization has approved.

Ability to easily change compile options for Development, Test and Production Builds	YES Meister allows the administrator to define the appropriate compile flags for each of the stages in the lifecycle. For example, a build for "PROD" would not include debug flags, but include optimization.	Ant - NO Ant allows you to code the script specifically for an environment adding whatever compile options are determined by the developer, even if this includes debug for a production build. This is done on a per script basis, not at a global level. Maven – Partial The Maven Compiler Plug-in allows you to define compiler options, but you can pass any compiler options during a build. There is no way to enforce a standard or prevent options from being passed.
Supports multiple tools and languages.	YES Meister can build different languages in a single build pass. As development languages grow more sophisticated there is a big push for cross-language integration. For instance, .NET applications working seamlessly with Java applications. Because Meister is language, operating system and platform independent, large companies can maintain a consistent build management process even though their choice of development languages and platforms change over time.	Partial for Both ANT and Maven were originally intended for building Jars, Wars and Ears. Since that time, enhancements have been made to support .Net (Nant) or the Eclipse C compiler. In addition, many smart programmers have tweeked their environments to support languages other than Java. Most of the features of both Ant and Maven are centered around the Java language and often is not a practical fit for .Net, C-Unix, Oracle Forms, etc. If an enterprise has requirements to build applications written in multiple languages, it is not uncommon that different build processes are required to satisfy each language. .
Auditing and Escrow	YES Meister includes the ability to create a Build Audit Report. A Build Audit Report shows every artifact used to create binaries, regardless of language or platform including, Jars, Wars, Ears, DLL, EXE, Unix binaries, etc. Going beyond a more traditional "Bill of Material" report, Meister includes in the Build Audit Report all artifacts used in the build, even when they did not come from the version control tool. When they do come from a version control tool, all item ...	NO for both Ant or Maven do not perform any detailed Auditing. Neither tool provides the ability to scan code and monitor files being passed to the compiler so creating a footprint that matches your executable is not available.

	... information is included. This report includes all artifacts coming from external repositories, on a local shared drive or simply referenced by the compiler. This information is used for auditing and escrow purposes. In addition the Build Audit Report can be embedded into the binary as a Footprint and can be viewed using "OMIDENT" at a command line. This makes it easy to quickly check who the artifact was created without referencing any other tool or database and guarantees source to binary integrity.	
Provides logging and reporting facilities to ensure an auditable build process.	YES Meister delivers both the ability to manage the compile and link process as well as a build management interface for tracking logs, trend analysis, and integration with other commercial and open source tools. - HTML Build Logs – Every Meister executed build can have an accompanying, detailed, web-based build log. The Build Logs can be used to validate each build process. - Build Audit Reports – Every Meister built component can have an accompanying Build Audit Report. Build Audit Reports detail version information for all source and non-source code dependencies used during a build. Build Audit reports ensure that an object in Production can always be rebuilt. - Impact Analysis (IA) Reports – HTML based logs are created that detail dependency relationships within built components. The IA Report can be used to evaluate the System wide impact of a coding change. - Trend Reports – Shows most frequent builds, slowest builds, fastest builds, etc. - ALM reports – Meister is integrated with common commercial and Open Source tools competitive to Jenkins/Hudson and displays the results of these reports in a common interface.	NO ANT has no built in logging or reporting functionality. All logging and reporting functionality needs to be scripted by developers in XML build scripts. Open Source tools such as Jenkins can manage the logs of Ant, but Ant itself does not provide these features.

Common build environment for the enterprise.	YES Often large enterprises will look to a centralized team to maintain their build process. Using Meister, team members responsible for builds of all types of languages can set build standards and maintain an ongoing build process. Instead of having multiple types of build scripts, reusable PERL scripts replace Ant, Maven and Make hard-coded scripts. Many organizations will move from hundreds of build scripts to a few dozen Meister Build Services.	NO Build maintenance and updates require users to understand the unique syntax of an ANT XML build scripts or Maven scripts or even make scripts. If problems occur when a centralized team is maintaining builds, the team members will often have to turn to Developers to fix the problem.

Conclusion

Ant/ Maven build and deploy scripts have their place as well as does OpenMake Meister. Organizations with smaller teams and those with less stringent process and audit requirements may be well served by Open Source solution using Ant/Maven and Jenkins. However, for larger enterprises that strive to carefully control and trace the build to deploy process across teams, an automated and auditable solution is required. OpenMake Meister meets these requirements by automatically generating the Ant based build.xml files providing a solid framework for enforcing standards and repeatability across the lifecycle. In addition Meister provides accurate incremental build and deploy processing for less risky releases, and delivers a framework for creating shared community developed knowledge simplifying software releases across diverse teams from Development through QA and Production control. Meister delivers a higher order of build management and automation that cannot be provided by Jenkins executing Ant and Maven based build and deploy scripts. From accelerating the software compiles, interrogating the artifacts for audit, and tightly controlling the use of the technology stack based on application life cycle stage, Meister offers features that address the entire organization while supporting the needs of developers for Continuous Integration, Agile methodologies and simplifying the hand-off the build and deploy process for testing and production control.

About OpenMake Software

OpenMake® Software, the DevOps Authority, delivers a dynamic solution for streamlining, accelerating, and standardizing build to deploy activities that can flex to meet your ever increasing operational demands. Our solutions automate tasks by eliminating script driven processes and static configurations while supporting continuous build and deploy. We enable you to manage incremental releases, leverage the cloud, increase productivity, eliminate bottlenecks, and provide management with actionable traceability reports. Over 400 companies worldwide use our solutions to dynamically align the development to release process from source to production.

Tracy Ragan – COO and Co-Founder, OpenMake Software

Ms. Ragan has had extensive experience in the development and implementation of business applications. It was during her consulting experiences that Ms. Ragan recognized the lack of build and release management procedures for the distributed platform that had long been considered standard on the mainframe and UNIX. In the four years leading to the creation of OpenMake Software she worked with development teams in implementing a team-centric standardized build to release process.
She can be reached at Tracy.Ragan@OpenMakesSoftware.com.

ASERVO Software

ASERVO Software GmbH

Konrad-Zuse-Platz 8 / 81829 München
Tel.: +49 (0) 89 7 16 71 82-40 / Fax: +49 (0) 89 7 16 71 82-55
Email: rmayr@aservo.com / www.ASERVO.com