Meister – Moving Beyond Jenkins

A summary whitepaper reviewing the differences between OpenMake Meister and Jenkins

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

OpenMake Software
www.openmakesoftware.com
312.440.9545 800.359.8049

Summary

Even though OpenMake Meister is a full DevOps solution, we are often asked how Meister is different from Jenkins. This question arises because both Meister and Jenkins are Continuous Integration servers with workflow management capabilities, centralized logging and server clustering. This is where their similarities end. Jenkins's main features are that of a job scheduler designed for the execution of build scripts such as Maven and Ant. In contrast to Jenkins, Meister fully automates the DevOps process and goes far beyond what simple job scheduling features can do by providing a standardized model driven framework for automating, standardizing and accelerating, without scripts, development operational tasks from Build to Deploy supporting multiple platforms and languages including Java, .Net, Oracle, C-Unix and others.

How they are the same

The feature that OpenMake Meister and Jenkins share relates to the Continuous Integration process. Continuous integration servers provide a basic workflow engine that can be triggered (based on source check-in and quiet period), scheduled or executed on demand. The workflow engine, like a job scheduler, executes a series of steps that can be distributed across a cluster of machines. Example steps would include calling a check-out from a versioning tool, executing a Maven script, calling static code analysis, executing a deploy and running tests. These steps can be distributed across multiple machines. The software build can be executed on a dedicated "build" server and testing can be executed on a dedicated "test" server. Output from each step in the process is centrally logged so everyone can easily review the results of all processes executed during the workflow.

Continuous Integration is what Jenkins offers. Continuous Integration is only a subset of what Meister offers.

How they are different

When using Jenkins, developers write and maintain scripts that perform the steps called by the continuous integration server. Most often, Java developers use Jenkins in conjunction with Maven build scripts.The core of the logic and automation for compiling code or executing a deployment is not done by Jenkins, and instead done by the scripts that Jenkins calls. For this reason, Jenkins can only be as smart as the scripts themselves. The most costly and error prone aspect of development operations is the writing and maintaining of these static scripts. In particular, build and deploy scripts can require frequent updates as source code and binaries change. Jenkins can call a Maven script, but cannot automatically repair the script if it has become out of date due to a simple coding update. Jenkins requires that a developer with knowledge of how the application is built and deployed serve as a middleman to make sure that the scripts execute correctly each time the continuous integration process is called.

Meister's Model Driven Framework addresses both continuous integration and the automation of build and deploys down to the lowest level of activity, without static, homegrown scripted processes. Meister automates the generation and management of build and deploys scripts providing a dynamic and intelligent DevOps platform. This allows developers to focus on coding core application components instead of writing and tweaking brittle scripts that are required for compiling code and deploying binaries. Like Jenkins, Meister allows you to control the high level activities around continuous integration such as calling static code analysis and testing in a defined order. Beyond what Jenkins can offer, Meister allows you to control the low level activities such as dynamically setting compiler options and controlling source code dependencies.

Because Meister executes builds and deploys with the full understanding of how the application is assembled and released, it can perform incremental builds and deploys, without special coding of scripts, or parallelize the process improving the performance of builds and deploys by as much as 50%. Meister supports a full Build Audit, showing precisely the files that were called by the compilers and linkers, integrated with versioning information. Meister can expose audit exceptions by showing all files that were used during the build, but not managed by a standard versioning process. This level of information is used for deployment ensuring that the libraries used during the creation of the binaries match what is running in production to minimize or completely eliminate cost of run time errors.

Meister also provides an integrated solution for managing server resources and clusters, including the ability to automatically provision and destruct virtual machines with its CloudBuilder option.

Jenkins is a popular tool for Java developers coding Maven scripts, but not so useful for the Microsoft Visual Studio .Net developer. In order for Jenkins to support a .Net build, Jenkins must be able to execute the .Net MSBuild process outside of the IDE. This is not a common practice in .Net development. The Jenkins user would recommend the use of Nant scripts to call MSBuild to execute the .Net Build. .Net users are accustomed to avoiding the manual scripting of the build process as .Net does this automatically for them for a single solution file, similar to Meister. Meister's continuous integration server integrates with MSBuild requiring no updates to the solution files or custom Nant scripts. Meister also provides features for compiling multiple .Net solution files as a single solution, a feature that is not offered by the Visual Studio platform alone. Meister integrates with TFS and TeamBuild to support multi-solution and multi-platform builds with continuous integration.

Below is a feature comparison of the similarities and differences between Meister and Jenkins:

Feature Description		
Continuous Integration	Meister	Jenkins
Built-in Continuous Integration Server	√	√
Check-in Changes monitored on Quite Period	√	√
Monitor of SCM repository for changes	√	√
Multiple CI configurations for one install or Instance	√	
Audit Trail of all Source and Compile Dependencies	√	
Impact Analysis of Source and Dependency Hierarchy	√	
Incremental build based on only changed source and dependencies	√	
Use of Approved Source and Compile Dependencies Only	√	
Automatic Download of Runtime Dependencies from Internet	√ (uses Archiva)	
Multiple Source and Compile Dependency Paths for same Project	√	
Centralized Reporting	Meister	Jenkins
Aggregation of Report Output to a single repository for all Platforms (UNIX, Windows, Linux, z/OS)	√	
Real-Time Monitoring of Build Results	√	
Build Acceleration	Meister	Jenkins
Parallelized Builds for improved build speeds	√	
Build Avoidance (Incremental Build processing)	√	
Workflow Management (Build Lifecycle)	Meister	Jenkins
Standard Build Loop support	√	√
Customizable Application Build Loop	√	√
Reusable Application Build Loops	√	
Distributed Remote Agents for distributed processing of Build Loop steps.	√	
Non-triggered Scheduling	√	
Out of the box Integration with commercial and Open Source ALM tools	√	√
Standards	Meister	Jenkins
Common Build Structure	√	
Centralized Management of Compile Options	√	
Build can be standardized to a custom structure	√	
Higher level of reusability between Java builds	√	
Higher level of reusability between C, C++ and .NET builds	√	
Easy compile ramp up for new Java developers	√	
Easy compile ramp up for new non-Java developers	√	
Calls Ant Tasks and Plug-ins	√	√

Distributed development	Meister	Jenkins
Server Pooling (Clusters)	√	√ (plug-in)
Centralized shared knowledge of all Build meta data	√	
Supports Cloud Provisioning	√	
IDE Support	Meister	Jenkins
External Build to Eclipse Integration	√	
Eclipse to External Build Integration	√	
Full Support of Eclipse with an Eclipse RCP	√	
.Net to External Build Integration	√	
Automatic generation of build scripts from Eclipse project data	√	
Automatic generation of build scripts from .	√	

Conclusion

Jenkins and Meister may look similar because of their continuous integration features. But at a closer look you will find that the work your developers do in writing and maintaining build to deploy scripts is eliminated by Meister reducing risk and cost. In addition, because Meister understands how your application is assembled through its scanning technology, it can offer improved build and deploy quality as well as improved and build and deploy speeds.

Meister manages your overall DevOps process with built-in intelligence eliminating the need for your developers to rely on a "team" build expert. It drastically reduces cost by automating, exposing and improving all aspects of the build to deploy process providing a model driven framework that is repeatable, auditable and 100% automated.

About OpenMake Software

OpenMake Software is the DevOps Authority providing an enterprise scale DevOpsframework for streamlining the build, test and deploy life cycle. Our solutions enable customers to reduce software development cycle times, improve communication between development and production control teams, increase developer productivity, and provide management with actionable audit and traceability reports. We go beyond what our competitors can offer by providing developers and production control a secure and dynamic operations framework that does not depend on brittle static scripts. Our solutions deliver on-demand cloud based server provisioning, environment configuration management, dependency management and continuous integration.
Over 100,000 users at 400 companies worldwide rely on the DevOps framework from OpenMake Software. For more information about OpenMake Meister or about OpenMake Software go to www.openmakesoftware.com or contact us at 312.440.9545, toll free 800.359.8049. You can email us at request-info@openmakesoftware.com