**Continuous integration with Jenkins**

**Setup a continuous integration process to build and test a Java web app with Jenkins and JUnit**

**Listing 1. Console output from a successful project build**

**Started** **by** user **Steven** **Haines**

**Building** **in** workspace /**Users**/shaines/.jenkins/workspace/hello-world-servlet

**Cloning** the remote **Git** repository

**Cloning** repository https:*//github.com/ligado/hello-world-servlet*

> git init /**Users**/shaines/.jenkins/workspace/hello-world-servlet *# timeout=10*

**Fetching** upstream changes **from** https:*//github.com/ligado/hello-world-servlet*

> git --version *# timeout=10*

> git -c core.askpass=**true** fetch --tags --progress https:*//github.com/ligado/hello-world-servlet +refs/heads/\*:refs/remotes/origin/\**

> git config remote.origin.url https:*//github.com/ligado/hello-world-servlet # timeout=10*

> git config --add remote.origin.fetch +refs/heads*/\*:refs/remotes/origin/\* # timeout=10*

*> git config remote.origin.url https://github.com/ligado/hello-world-servlet # timeout=10*

*Fetching upstream changes from https://github.com/ligado/hello-world-servlet*

*> git -c core.askpass=true fetch --tags --progress https://github.com/ligado/hello-world-servlet +refs/heads/\*:refs/remotes/origin/\**

*> git rev-parse refs/remotes/origin/master^{commit} # timeout=10*

*> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10*

*Checking out Revision a3da5d8d82d11475e5e1fc899871f1f7a975ed07 (refs/remotes/origin/master)*

*> git config core.sparsecheckout # timeout=10*

*> git checkout -f a3da5d8d82d11475e5e1fc899871f1f7a975ed07*

*First time build. Skipping changelog.*

*Unpacking https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.3.9/apache-maven-3.3.9-bin.zip to /Users/shaines/.jenkins/tools/hudson.tasks.Maven\_MavenInstallation/Maven\_3.3.9 on Jenkins*

*[hello-world-servlet] $ /Users/shaines/.jenkins/tools/hudson.tasks.Maven\_MavenInstallation/Maven\_3.3.9/bin/mvn clean install*

*[INFO] Scanning for projects...*

*[WARNING]*

*[WARNING] Some problems were encountered while building the effective model for com.geekcap.vmturbo:hello-world-servlet-example:war:1.0-SNAPSHOT*

*[WARNING] 'dependencies.dependency.(groupId:artifactId:type:classifier)' must be unique: junit:junit:jar -> version 3.8.1 vs 4.12 @ line 23, column 19*

*[WARNING]*

*[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.*

*[WARNING]*

*[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.*

*[WARNING]*

*[INFO]*

*[INFO] ------------------------------------------------------------------------*

*[INFO] Building hello-world-servlet-example Maven Webapp 1.0-SNAPSHOT*

*[INFO] ------------------------------------------------------------------------*

*[INFO]*

*[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ hello-world-servlet-example ---*

*[INFO]*

*[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ hello-world-servlet-example ---*

*[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!*

*[INFO] skip non existing resourceDirectory /Users/shaines/.jenkins/workspace/hello-world-servlet/src/main/resources*

*[INFO]*

*[INFO] --- maven-compiler-plugin:2.0.2:compile (default-compile) @ hello-world-servlet-example ---*

*[INFO] Compiling 2 source files to /Users/shaines/.jenkins/workspace/hello-world-servlet/target/classes*

*[INFO]*

*[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ hello-world-servlet-example ---*

*[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!*

*[INFO] skip non existing resourceDirectory /Users/shaines/.jenkins/workspace/hello-world-servlet/src/test/resources*

*[INFO]*

*[INFO] --- maven-compiler-plugin:2.0.2:testCompile (default-testCompile) @ hello-world-servlet-example ---*

*[INFO] Compiling 1 source file to /Users/shaines/.jenkins/workspace/hello-world-servlet/target/test-classes*

*[INFO]*

*[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ hello-world-servlet-example ---*

*[INFO] Surefire report directory: /Users/shaines/.jenkins/workspace/hello-world-servlet/target/surefire-reports*

*-------------------------------------------------------*

*T E S T S*

*-------------------------------------------------------*

*Running com.geekcap.vmturbo.ThingTest*

*Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.05 sec*

*Results :*

*Tests run: 1, Failures: 0, Errors: 0, Skipped: 0*

*[INFO]*

*[INFO] --- maven-war-plugin:2.2:war (default-war) @ hello-world-servlet-example ---*

*[INFO] Packaging webapp*

*[INFO] Assembling webapp [hello-world-servlet-example] in [/Users/shaines/.jenkins/workspace/hello-world-servlet/target/helloworld]*

*[INFO] Processing war project*

*[INFO] Copying webapp resources [/Users/shaines/.jenkins/workspace/hello-world-servlet/src/main/webapp]*

*[INFO] Webapp assembled in [61 msecs]*

*[INFO] Building war: /Users/shaines/.jenkins/workspace/hello-world-servlet/target/helloworld.war*

*[INFO] WEB-INF/web.xml already added, skipping*

*[INFO]*

*[INFO] --- maven-install-plugin:2.4:install (default-install) @ hello-world-servlet-example ---*

*[INFO] Installing /Users/shaines/.jenkins/workspace/hello-world-servlet/target/helloworld.war to /Users/shaines/.m2/repository/com/geekcap/vmturbo/hello-world-servlet-example/1.0-SNAPSHOT/hello-world-servlet-example-1.0-SNAPSHOT.war*

*[INFO] Installing /Users/shaines/.jenkins/workspace/hello-world-servlet/pom.xml to /Users/shaines/.m2/repository/com/geekcap/vmturbo/hello-world-servlet-example/1.0-SNAPSHOT/hello-world-servlet-example-1.0-SNAPSHOT.pom*

*[INFO] ------------------------------------------------------------------------*

*[INFO] BUILD SUCCESS*

*[INFO] ------------------------------------------------------------------------*

*[INFO] Total time: 2.932 s*

*[INFO] Finished at: 2016-08-20T06:35:54-04:00*

*[INFO] Final Memory: 23M/316M*

*[INFO] ------------------------------------------------------------------------*

*Finished: SUCCESS*

From the output shown in Listing 1, the first thing that Jenkins does is check out the source code from GitHub to the directory /Users/shaines/.jenkins/workspace/hello-world-servlet.

You can open a terminal window and navigate to that directory to confirm. In order to check out your project, you can see several command-line Git statements: git init to setup the project, git config to setup the remote repository references, and git checkout to retrieve the source code.

Next, because this is the first time you've run the project build and you didn't already have Maven installed, Jenkins downloaded Maven from Apache (you can see the URL in the console output) and decompressed it to /Users/shaines/.jenkins/tools/hudson.tasks.Maven\_MavenInstallation/Maven\_3.3.9. When we get deeper into Maven, you'll be able to see more of the command-line interface tools installed under the tools folder.

Finally, Jenkins executed your Maven goals with the following command:

/**Users**/shaines/.jenkins/tools/hudson.tasks.**Maven\_MavenInstallation**/**Maven\_3**.3.9/bin/mvn clean install

In Listing 1, you can see the source getting built, the unit tests executing, and finally Maven publishing the build artifact (helloworld.war) to a local machine's local Maven repository.

**Troubleshooting build failures in Jenkins**

What if a build fails? In that case you can retrace your steps to troubleshoot the failure. Just click on the failed build and choose **Console Output**. As you can see from the console output from my first failed build, I initially forgot to install Maven.

[[](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig08-copy-100820970-orig.jpg)](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig08-copy-100820970-orig.jpg" \o "<div class='credit'>Steven Haines</div> <p>Figure 8.&nbsp;A failed build alert in Jenkins</p> )

Figure 8. A failed build alert in Jenkins

Fortunately, I received an error message that Jenkins couldn't run Maven (mvn), which alerted me to my oversight!

TABLE OF CONTENTS

* [Introduction to CI/CD](https://www.javaworld.com/article/3123117/open-source-java-projects-jenkins-with-docker-part-1.html?page=1#toc-1)
* [Download and install Jenkins](https://www.javaworld.com/article/3123117/open-source-java-projects-jenkins-with-docker-part-1.html?page=1#toc-2)
* [Setup Jenkins CI for a Java web app](https://www.javaworld.com/article/3123117/open-source-java-projects-jenkins-with-docker-part-1.html?page=1#toc-3)
* [CI reports with Jenkins and JUnit](https://www.javaworld.com/article/3123117/open-source-java-projects-jenkins-with-docker-part-1.html?page=2#toc-4)
* [Using static code analysis tools with Jenkins](https://www.javaworld.com/article/3123117/open-source-java-projects-jenkins-with-docker-part-1.html?page=2#toc-5)

**CI reports with Jenkins and JUnit**

Once you have a working build, you'll want to start gathering information into a set of reports. In the previous section, when you executed mvn clean install, the install goal implicitly ran your unit tests. Unit tests are finely grained tests that validate your application from a class and method perspective. In this section we'll configure Jenkins to generate a report that summarizes the unit test execution. Then I'll show you how to run and report static code analysis reports for things like test coverage, code complexity, and detecting common bugs.

**JUnit test report**

Open your **hello-world-servlet** project and click **Configure**. Scroll down to the bottom of the page, find the Post-build Actions section, and click **Add post-build action** and choose Publish JUnit test result report. Enter the following in the Test Reports XMLs text field:

\*\*/target/surefire-reports*/\*.xml*

The Test Reports XMLs field points to the directory where your Surefire tests are published. Maven creates a target directory, a surefire-reports subdirectory, and then publishes a set of XML files summarizing the test results to the new directory. You can navigate to the directory yourself to confirm:

~/.jenkins/workspace/hello-world-servlet/target/surefire-reports

Because this is a simple example with only one test in it, Maven creates a single XML file named TEST-com.geekcap.vmturbo.ThingTest.xml. In a production application you would have dozens or even hundreds of XML files.

When you're finished, press **Save**. You should see something similar to Figure 9.

[[](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig09-copy-100820972-orig.jpg)](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig09-copy-100820972-orig.jpg" \o "<div class='credit'>Steven Haines</div> <p>Figure 9.&nbsp;Adding a JUnit report post-build action</p> )

Figure 9. Adding a JUnit report post-build action

From your project page, press the **Build Now** button. When that's complete, click on the new build. You should see a new section at the bottom of the page that reads Test Results (no failures). Click on the link and you should be presented with a screen similar to Figure 10.

[[jw jenkins fig10 copy](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig10-copy-100820973-orig.jpg)](https://images.idgesg.net/images/article/2019/12/jw-jenkins-fig10-copy-100820973-orig.jpg" \o "<div class='credit'>Steven Haines</div> <p>Figure 10.&nbsp;A JUnit report test summary</p> )

Figure 10. A JUnit report test summary

This report shows a summary of all passes and failures and allows you to drill down into packages, classes, and individual tests to see the results.

**The JUnit Plugin**

If you're curious about why the new report is available in your new builds, but not retroactively available in previous builds, you'll need to dive further into the inner workings of Jenkins, and specifically of the JUnit plugin.

When you configured a post-build action to get the JUnit test report, you invoked the [JUnit Plugin](https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin). When a build is complete, Jenkins creates a directory for it in its .jenkins/jobs/builds folder. You can navigate to that folder to see all of your build artifacts. Here's the difference that I have between build "1" and build "2" (the before and after for the JUnit report):

$ ls -l 1

total 32

-rw-r----- 1 shaines staff 2341 **Aug** 20 06:35 build.xml

-rw-r----- 1 shaines staff 0 **Aug** 20 06:35 changelog.xml

-rw-r----- 1 shaines staff 8748 **Aug** 20 06:35 log

$ ls -l 2

total 40

-rw-r----- 1 shaines staff 2658 **Aug** 21 16:46 build.xml

-rw-r----- 1 shaines staff 0 **Aug** 21 16:46 changelog.xml

-rw-r----- 1 shaines staff 707 **Aug** 21 16:46 junitResult.xml

-rw-r----- 1 shaines staff 8209 **Aug** 21 16:46 log

The build directory contains a build.xml file that specifies the configuration that was in place when that build was executed, a changelog.xml file, and a log file that saves the console output from each build. In directory 2 you can see a new junitResult.xml file that contains your results. The JUnit Plugin provides a stylesheet to render the XML file in a more readable fashion.

**Using static code analysis tools with Jenkins**

Static code analysis uses various standard measures to test the overall health of your application. In Jenkins, you can combine several good static code analysis tools for an assessment. In this case, we'll use the following tools and generate a report of their findings:

* [Checkstyle](https://maven.apache.org/plugins/maven-checkstyle-plugin/) checks the style of your code against the official Java code style.
* [FindBugs](http://gleclaire.github.io/findbugs-maven-plugin/) searches for and reports common bugs.
* [PMD](https://maven.apache.org/plugins/maven-pmd-plugin/) validates build rules.
* [Cobertura](http://www.mojohaus.org/cobertura-maven-plugin/) checks the coverage of your unit tests.

You need to do the following in order to use these tools and generate reports for them in Jenkins:

1. Add reporting plugins to your Maven pom.xml file.
2. Install the Jenkins plugins for these tools.
3. Update your Maven goals to call these plugins.
4. Add post-build actions to publish the static code analysis reports.

**Step 1. Add the reporting plugins to your Maven pom.xml**

I updated the POM file locally and pushed it to my GitHub with sample configurations that define Maven plugins for these tools. Listing 2 is for your reference.

**Listing 2. pom.xml**

**<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"**>**

**<modelVersion>**4.0.0**</modelVersion>**

**<groupId>**com.geekcap.vmturbo**</groupId>**

**<artifactId>**hello-world-servlet-example**</artifactId>**

**<packaging>**war**</packaging>**

**<version>**1.0-SNAPSHOT**</version>**

**<name>**hello-world-servlet-example Maven Webapp**</name>**

**<url>**http://maven.apache.org**</url>**

**<reporting>**

**<plugins>**

**<plugin>**

**<groupId>**org.codehaus.mojo**</groupId>**

**<artifactId>**cobertura-maven-plugin**</artifactId>**

**<version>**2.7**</version>**

**<configuration>**

**<formats>**

**<format>**xml**</format>**

**</formats>**

**</configuration>**

**</plugin>**

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-checkstyle-plugin**</artifactId>**

**<version>**2.17**</version>**

**</plugin>**

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-pmd-plugin**</artifactId>**

**<version>**3.6**</version>**

**<configuration>**

**<linkXref>**true**</linkXref>**

**<sourceEncoding>**utf-8**</sourceEncoding>**

**<minimumTokens>**100**</minimumTokens>**

**<targetJdk>**1.6**</targetJdk>**

*<!--*

*<rulesets>*

*<ruleset>/rulesets/braces.xml</ruleset>*

*</rulesets>*

*<excludes>*

*<exclude>\*\*/\*Bean.java</exclude>*

*<exclude>\*\*/generated/\*.java</exclude>*

*</excludes>*

*<excludeRoots>*

*<excludeRoot>target/generated-sources/stubs</excludeRoot>*

*</excludeRoots>*

*-->*

**</configuration>**

**</plugin>**

**<plugin>**

**<groupId>**org.codehaus.mojo**</groupId>**

**<artifactId>**taglist-maven-plugin**</artifactId>**

**<version>**2.3**</version>**

*<!--*

*<configuration>*

*<tags>*

*<tag>TODO</tag>*

*<tag><todo</tag>*

*<tag>FIXME</tag>*

*<tag>DOCUMENT\_ME</tag>*

*<tag>NOT\_YET\_DOCUMENTED</tag>*

*</tags>*

*</configuration>*

*-->*

**</plugin>**

**<plugin>**

**<groupId>**org.codehaus.mojo**</groupId>**

**<artifactId>**findbugs-maven-plugin**</artifactId>**

**<version>**3.0.4**</version>**

**<configuration>**

**<effort>**Max**</effort>**

**</configuration>**

**</plugin>**

**</plugins>**

**</reporting>**

**<dependencies>**

**<dependency>**

**<groupId>**javax.servlet**</groupId>**

**<artifactId>**servlet-api**</artifactId>**

**<version>**2.5**</version>**

**<scope>**provided**</scope>**

**</dependency>**

**<dependency>**

**<groupId>**junit**</groupId>**

**<artifactId>**junit**</artifactId>**

**<version>**4.12**</version>**

**<scope>**test**</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<finalName>**helloworld**</finalName>**

**<plugins>**

**<plugin>**

**<groupId>**org.apache.maven.plugins**</groupId>**

**<artifactId>**maven-compiler-plugin**</artifactId>**

**<version>**2.0.2**</version>**

**<configuration>**

**<source>**1.6**</source>**

**<target>**1.6**</target>**

**</configuration>**

**</plugin>**

**<plugin>**

**<groupId>**org.jolokia**</groupId>**

**<artifactId>**docker-maven-plugin**</artifactId>**

**<version>**0.13.4**</version>**

**<configuration>**

**<dockerHost>**tcp://192.168.99.100:2376**</dockerHost>**

**<certPath>**/Users/shaines/.docker/machine/machines/default**</certPath>**

**<useColor>**true**</useColor>**

**<images>**

**<image>**

**<name>**lygado/tomcat-with-my-app:0.1**</name>**

**<alias>**service**</alias>**

**<build>**

**<from>**tomcat**</from>**

*<!--*

*<ports>*

*<port>8080</port>*

*</ports>*

*<entryPoint>*

*<exec>*

*<args>catalina.sh</args>*

*<args>start</args>*

*</exec>*

*</entryPoint>*

*-->*

**<assembly>**

**<mode>**dir**</mode>**

**<basedir>**/usr/local/tomcat/webapps**</basedir>**

**<descriptor>**assembly.xml**</descriptor>**

**</assembly>**

**</build>**

**<run>**

**<ports>**

**<port>**8080:8080**</port>**

**</ports>**

**</run>**

**</image>**

**</images>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

The reporting section adds the four aforementioned static code analysis plugins. Each of these plugins defines Maven goals that can be invoked as follows to generate their respective reports:

mvn clean install checkstyle:checkstyle findbugs:findbugs pmd:pmd pmd:cpd cobertura:cobertura -**Dcobertura**.report.format=xml

**Step 2. Install the Jenkins plugins**

Next, you'll need to install the Jenkins plugins that will retrieve the generated reports, associate them with a build, and make them available through the Jenkins dashboard.