

Sonatype CLM for Maven

Contents

1	Introduction	1
2	Creating a Component Index	3
2.1	Excluding Module Information Files in Continuous Integration Tools	4
3	Creating a Component Info Archive for Nexus Pro CLM Edition	6
4	Evaluating Project Components with Sonatype CLM Server	8
5	Simplifying Command Line Invocations	11
6	Skipping Executions	12
7	Using Sonatype CLM for Maven with Other IDEs	13
7.1	Maven Plugin Setup	13
7.2	IntelliJ IDEA	14

7.3 NetBeans IDE	16
----------------------------	----

8 Conclusion	19
---------------------	-----------

List of Figures

7.1	Creating a Maven Run Configuration for a CLM Evaluation in IntelliJ	15
7.2	Maven Projects View with the CLM Evaluation Run Configuration in IntelliJ	15
7.3	CLM for Maven Output in the Run Console in IntelliJ	16
7.4	Project View with the <code>pom.xml</code> in NetBeans	17
7.5	Maven Goal Setup for a CLM Evaluation in NetBeans	17
7.6	CLM for Maven Output in the Output Window in NetBeans	18

Chapter 1

Introduction

Sonatype CLM for Maven allows users to evaluate any Maven-based software projects, in the same way our integrated tools (e.g. Nexus Pro, Eclipse, Hudson/Jenkins) do.

This means you can access the same robust, reporting features no matter what toolset you use. It can be run on a command line interface and can therefore be executed on any continuous integration server, as well as a number of popular IDEs.

How you use the Sonatype CLM for Maven plugin widely depends on how you enforce policy. However, in general, when using the plugin on a multi-module project in most cases you will only configure an execution for the modules that produce components that will be deployed as an application.

Typically these are `ear` files or `war` files for deployment on application servers or `tar.gz` or other archives that are used for production deployments or distribution to users. That said, you can also analyze all modules of a project. Again, this will largely depend on what your CLM policy is enforcing and what you want to validate.

In the following sections, we'll provide details on these goals and their usage:

Index

The `index` goal of the plugin allows you to prepare data for analysis with Sonatype CLM for CI.

Attach

The `attach` goal aids your integration with Sonatype Nexus CLM Edition and the release process

using the staging tools of Nexus.

Evaluate

The `evaluate` goal can trigger an evaluation directly against a Sonatype CLM server.

Note

The `help` goal provides documentation for all the goals and parameters and you can invoke it with an execution like `mvn com.sonatype.clm:clm-maven-plugin:help`

Chapter 2

Creating a Component Index

When evaluating a Maven-based software project, Sonatype CLM for Maven can take advantage of the dependency information contained in the project's `pom.xml` files and the information about transitive dependencies available to Maven.

The `index` goal of Sonatype CLM for Maven allows you to identify component dependencies and makes this information available to Sonatype CLM CI tools (e.g. Sonatype CLM for Hudson/Jenkins or Bamboo). You can invoke an execution of the `index` goal manually as part of your command line invocation by executing the `index` goal after the `package` phase:

```
mvn clean install com.sonatype.clm:clm-maven-plugin:index
```

Alternatively you can configure the execution in the `pom.xml` files `build` section or in a profile's `build` section.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.sonatype.clm</groupId>
      <artifactId>clm-maven-plugin</artifactId>
      <version>2.1.1</version>
      <executions>
        <execution>
          <goals>
            <goal>index</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```
    </executions>
  </plugin>
</plugins>
</build>
```

With the above configuration a normal Maven build execution with e.g. `mvn clean install` will trigger the CLM plugin to be executed in the `package` phase and result in a log output similar to

```
[INFO] --- clm-maven-plugin:2.1.1:index (default) @ test-app ---
[INFO] Saved module information to /opt/test-app/target/sonatype-clm/ ↔
      module.xml
```

If you want to manually configure the lifecycle phase to execute the plugin, you have to choose a phase after `package`.

The generated `module.xml` file contains the information that will be picked up by Sonatype CLM for CI and incorporated into the CLM evaluation. This improves the analysis since Sonatype CLM for Maven is able to create a complete dependency list rather than relying on binary build artifacts.

Note

By default only dependencies in the `compile` and `runtime` scopes will be considered, since this reflects what other Maven packaging plugins typically include. Dependencies with the scopes `test`, `provided` and `system` must be manually added, and are described in the [Evaluating Project Components with Sonatype CLM Server section](#).

2.1 Excluding Module Information Files in Continuous Integration Tools

When using the Sonatype CLM Maven plugin and the `index` goal, module information files are created. If desired, you can exclude some of the modules from being evaluated. For example, you may want to exclude modules that support your tests, and don't contribute to the distributed application binary.

The default location where the module information files are stored is `${project.build.directory}/sonatype`

In the supported CI tool, you will see a section labeled *Module Excludes*. On this area, use a comma-separated list of [Apache Ant styled patterns](#) relative to the workspace root that denote the module infor-

mation files (`**/sonatype-clm/module.xml`) to be ignored.

Here's an example of the pattern described above:

```
**/my-module/target/**, **/another-module/target/**
```

If unspecified, all modules will contribute dependency information (if any) to the evaluation.

Chapter 3

Creating a Component Info Archive for Nexus Pro CLM Edition

The `attach` goal scans the dependencies and build artifacts of a project and attaches the results to the project as another artifact in the form of a `scan.xml.gz` file. It contains all the checksums for the dependencies and their classes and further meta information and can be found in the `target/sonatype-clm` directory. A separate `scan.xml.gz` file is generated for each maven module in an aggregator project in which the plugin is executed.

This attachment causes the file to be part of any Maven `install` and `deploy` invocation. When the deployment is executed against a Sonatype Nexus CLM Edition server the artifact is used to evaluate policies against the components included in the evaluation.

To use this goal, add an execution for it in the POM, e.g. as part of a profile used during releases:

```
<build>
  <plugins>
    <plugin>
      <groupId>com.sonatype.clm</groupId>
      <artifactId>clm-maven-plugin</artifactId>
      <version>2.1.1</version>
      <executions>
        <execution>
          <goals>
            <goal>attach</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Once configured in your project, the build log will contain messages similar to

```
[INFO] --- clm-maven-plugin:2.1.1:attach (default) @ test-app ---
[INFO] Starting scan...
[INFO] Scanning ...plexus-utils-3.0.jar
[INFO] Scanning ...maven-settings-3.0.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /opt/test-app/target/sonatype-clm/scan.xml.gz
```

The attachment of the `scan.xml.gz` file as a build artifact causes it to be stored in the local repository as well as the deployment repository manager or the Nexus staging repository ending with `-sonatype-clm-scan.xml.gz`. This file will be picked up by Sonatype Nexus CLM Edition and used in the policy analysis during the staging process. It improves the analysis since Sonatype CLM for Maven is able to create a complete dependency list rather than relying on binary build artifacts.

Chapter 4

Evaluating Project Components with Sonatype CLM Server

The `evaluate` goal scans the dependencies and build artifacts of a project and directly submits the information to a Sonatype CLM Server for policy evaluation.

If a policy violation is found and the CLM stage is configured to `Fail`, the Maven build will fail. If invoked for an aggregator project, dependencies of all child modules will be considered.

The `evaluate` goal requires the Sonatype CLM Server URL as well as the application identifier to be configured. Optionally a CLM stage can be configured.

The command line arguments are

`clm.serverUrl`

the URL for the CLM server, this parameter is required

`clm.applicationId`

the application identifier for the application to run policy against, this parameter is required

`clm.resultFile`

the path for specifying the location of a JSON file where the following information will be stored:

- `applicationId` : Application ID

- scanId : Organization ID
- reportHtmlUrl : URL to the HTML version of the report
- reportPdfUrl : URL to the PDF version of the report
- reportDataUrl : URL to the Data version of the report (for use via CURL, or similar tool)

clm.stage

the stage to run policy against with the possible values of develop, build, stage-release, release and operate with a default value of build.

clm.additionalScopes

the additional scopes you would like CLM to include components from during the evaluation. Values include test, provided, and system. In cases where you want to include more than one of these, separate the list using a comma (see examples below).

An example invocation is:

```
mvn com.sonatype.clm:clm-maven-plugin:evaluate -Dclm.additionalScopes=test ↵  
,provided,system -Dclm.applicationId=test -Dclm.serverUrl=http:// ↵  
localhost:8070
```

You can avoid specifying the parameters on the command line by adding them to your settings.xml or pom.xml as properties.

```
<properties>  
  <clm.serverUrl>http://localhost:8070</clm.serverUrl>  
  <clm.applicationId>test</clm.applicationId>  
</properties>
```

Alternatively the invocation can be configured in a pom.xml file:

```
<build>  
  <plugins>  
    <plugin>  
      <groupId>com.sonatype.clm</groupId>  
      <artifactId>clm-maven-plugin</artifactId>  
      <version>2.1.1</version>  
      <executions>  
        <execution>  
          <goals>  
            <goal>evaluate</goal>  
          </goals>  
          <phase>package</phase>  
          <configuration>  
            <serverUrl>http://localhost:8070</serverUrl>
```

```
        <stage>build</stage>
        <applicationId>test</applicationId>
        <additionalScopes>test,provided,system</additionalScopes>
    </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Sonatype CLM for Maven can be executed against an aggregator project. When executed in an aggregator project, it calculates the dependencies and transitive dependencies of all child modules and takes all of them into account for the policy evaluation. It is advisable to set the `inherited` flag for the plugin to `false` to avoid duplicate runs of the plugin in each module.

**Caution**

When bound to a lifecycle in a multi-module build, the plugin will take all dependencies of the Maven reactor into consideration for its analysis and not just the dependencies of the current module.

The `evaluate` goal logs its activity and provides the location of the generated report.

```
[INFO] --- clm-maven-plugin:2.1.1:evaluate (default) @ test-app ---
[WARNING] Goal 'evaluate' is not expected to be used as part of project lifecycle.
[INFO] Starting scan...
[INFO] Scanning ../repository/org/codehaus/plexus/plexus-utils/3.0/plexus- ←
utils-3.0.jar...
[INFO] Scanning ../repository/org/apache/maven/maven-settings/3.0/maven- ←
settings-3.0.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /opt/test-app/target/sonatype-clm/scan.xml.gz
[INFO] Uploading scan to http://localhost:8070 ...
[INFO] Evaluating policies... (ETA 5s)
[INFO] Policy Action: None
Summary of policy violations: 0 critical, 0 severe, 0 moderate
The detailed report can be viewed online at
http://localhost:8070/ui/links/application/test/report/f4582a1570634dc2ac8
```

After a successful build the report can be accessed in the Sonatype CLM server under the application that was configured. A direct link is provided on the log.

Chapter 5

Simplifying Command Line Invocations

If you happen to use the plugin frequently by running it manually on the command line and want to shorten the command line even more, you can add a plugin group entry to your Maven `settings.xml` file:

```
<settings>
...
<pluginGroups>
  <pluginGroup>com.sonatype.clm</pluginGroup>
  ...
</pluginGroups>
...
</settings>
```

This enables you to invoke the plugin using its shorthand prefix form:

```
mvn ... clm:index
```

Chapter 6

Skipping Executions

The `clm.skip` parameter can be used, when a CLM plugin execution is configured in your project's `pom.xml` file, but you want to avoid the execution for a particular build. An example execution is

```
mvn clean install -Dclm.skip=true
```

The parameter can also be set in your IDE configuration for Maven build executions or as a property in your `settings.xml` or `pom.xml`:

```
<properties>  
  <clm.skip>true</clm.serverUrl>  
</properties>
```


Chapter 7

Using Sonatype CLM for Maven with Other IDEs

While the integration with Eclipse offered by Sonatype CLM for IDE is the most powerful tooling for developers available, user of other popular integrated development environments are not left without support. All common Java IDEs have powerful integration with Apache Maven and therefore can be used together with Sonatype CLM for Maven to evaluate projects against your Sonatype CLM server.

This chapter showcases the integration with [IntelliJ IDEA from JetBrains](#) and [NetBeans IDE from Oracle](#).

7.1 Maven Plugin Setup

In our example setup for the usage with other IDE's we are going to add a plugin configuration for Sonatype CLM for Maven into the `pom.xml` file of the project we want to analyze as documented in [Example Configuration of Sonatype CLM for Maven](#). This configuration defines the `serverUrl` of the CLM server to be contacted for the evaluation, the `applicationId` used to identify the application in the CLM server to evaluate against and the `stage` configuration to use for the evaluation.

Example Configuration of Sonatype CLM for Maven

```
<build>
  <pluginManagement>
```

```
<plugins>
  <plugin>
    <groupId>com.sonatype.clm</groupId>
    <artifactId>clm-maven-plugin</artifactId>
    <version>2.1.1</version>
    <configuration>
      <serverUrl>http://localhost:8070</serverUrl>
      <applicationId>test</applicationId>
      <stage>develop</stage>
    </configuration>
  </plugin>
</plugins>
</pluginManagement>
</build>
```

With this configuration in place a user can kick off an evaluation with the command line `mvn package clm:evaluate`.

This will result in an output detailing the components to be analyzed, any policy violations and a link to the resulting report in the Sonatype CLM server.

Note

To speed the build up you can skip the test compilation and execution by passing `-Dmaven.test.skip=true` on the command line invocation, since it is not needed for the CLM evaluation.

7.2 IntelliJ IDEA

IntelliJ IDEA supports Maven projects natively and you can simply open a project in the IDE by opening the `pom.xml` file.

Once your project is opened and you have added the plugin configuration for Sonatype CLM for Maven from [Example Configuration of Sonatype CLM for Maven](#), you can create a configuration to run the desired Maven command.

Select *Edit Configurations* from the *Run* menu, press the + button and select *Maven* to add a new configuration. Enter the command line and other desired details as displayed in [Figure 7.1](#)

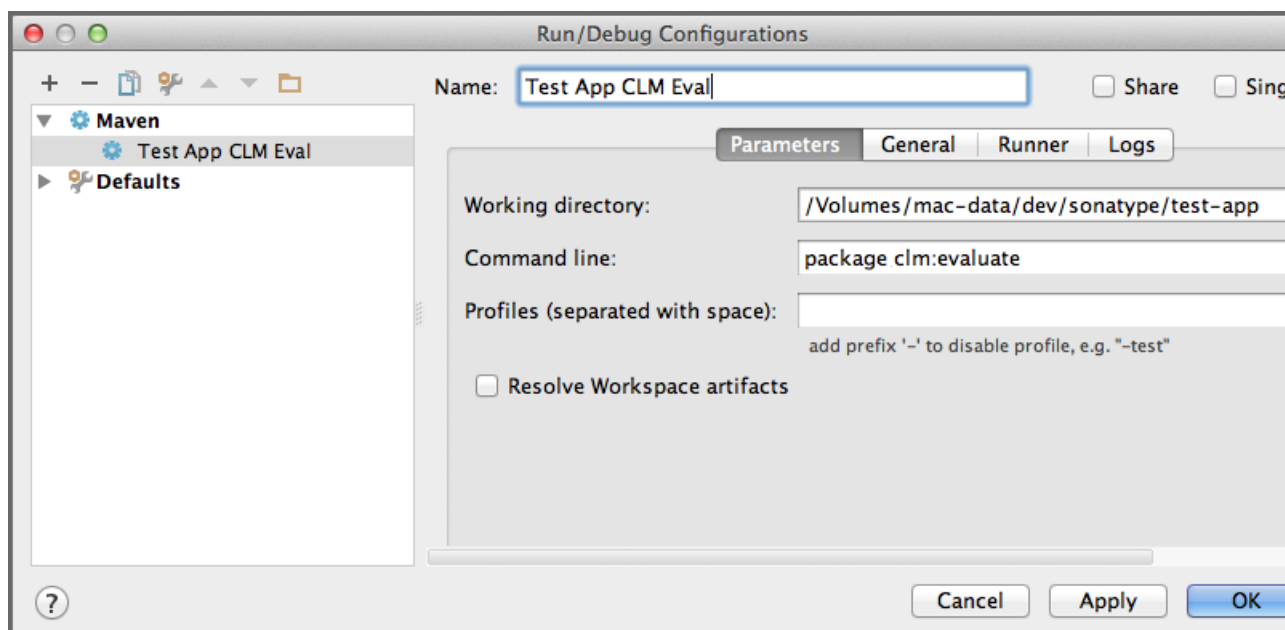


Figure 7.1: Creating a Maven Run Configuration for a CLM Evaluation in IntelliJ

After pressing *OK* in the dialog the new configuration will be available in the run configuration drop down as well the *Maven Projects* view. You can open the view using the *View* menu, selecting *Tools Window* and pressing *Maven Projects*. You will see the window appear in the IDE looking similar to Figure 7.2. It displays the run configuration selector with the green play button on the top as well as the Maven project with the CLM evaluation run configuration.

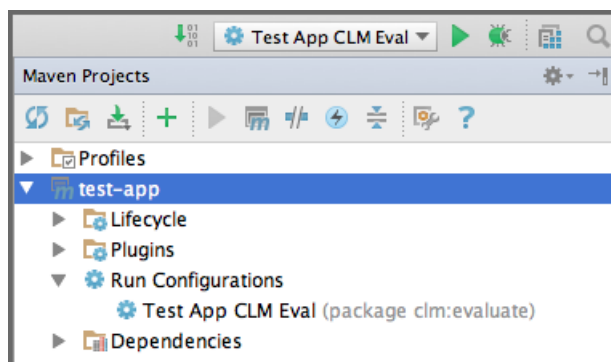
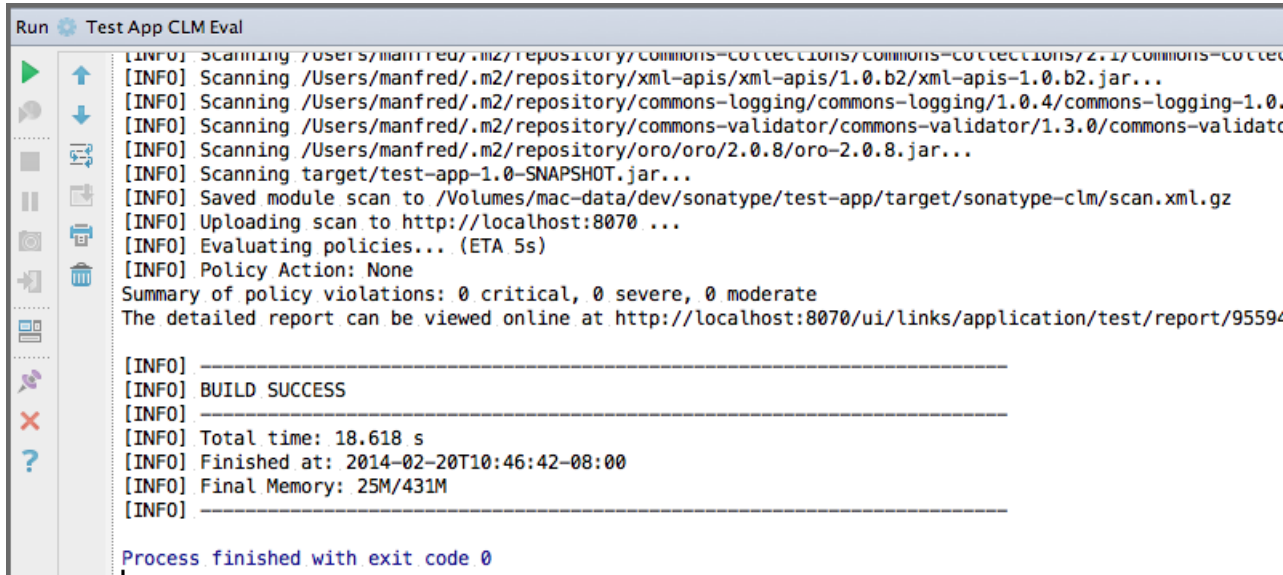


Figure 7.2: Maven Projects View with the CLM Evaluation Run Configuration in IntelliJ

You can press the green play button beside the run configuration, or the configuration entry itself in the *Maven Projects* window, to start a build. The build will run in an embedded console window in the IDE as displayed in Figure 7.3, and show all the output from the Maven build including any error messages and the link to the produced report in the Sonatype CLM server. Policy violations can be configured to result in a build failure.

The image shows a screenshot of the IntelliJ Run Console window. The title bar reads "Run" and "Test App CLM Eval". On the left side, there is a vertical toolbar with icons for running (a green play button), debugging (a blue bug), and other IDE functions. The main area of the console displays the output of a Maven build with Sonatype CLM integration. The output includes several "[INFO]" messages: scanning various JARs from the local repository (commons-collections, xml-apis, commons-logging, commons-validator, oro), scanning the target JAR (test-app-1.0-SNAPSHOT.jar), saving the module scan to a local file, uploading the scan to a local endpoint (http://localhost:8070), evaluating policies (ETA 5s), and a policy action of "None". A summary of policy violations shows 0 critical, 0 severe, and 0 moderate violations. A link to the detailed report is provided: http://localhost:8070/ui/links/application/test/report/9559. The output concludes with "BUILD SUCCESS" and a summary of the build: Total time: 18.618 s, Finished at: 2014-02-20T10:46:42-08:00, and Final Memory: 25M/431M. At the bottom, a status message says "Process finished with exit code 0".

```
[INFO] Scanning /Users/manfred/.m2/repository/commons-collections/commons-collections/2.1/commons-collections-2.1.jar...
[INFO] Scanning /Users/manfred/.m2/repository/xml-apis/xml-apis/1.0.b2/xml-apis-1.0.b2.jar...
[INFO] Scanning /Users/manfred/.m2/repository/commons-logging/commons-logging/1.0.4/commons-logging-1.0.4.jar...
[INFO] Scanning /Users/manfred/.m2/repository/commons-validator/commons-validator/1.3.0/commons-validator-1.3.0.jar...
[INFO] Scanning /Users/manfred/.m2/repository/oro/oro/2.0.8/oro-2.0.8.jar...
[INFO] Scanning target/test-app-1.0-SNAPSHOT.jar...
[INFO] Saved module scan to /Volumes/mac-data/dev/sonatype/test-app/target/sonatype-clm/scan.xml.gz
[INFO] Uploading scan to http://localhost:8070 ...
[INFO] Evaluating policies... (ETA 5s)
[INFO] Policy Action: None
Summary of policy violations: 0 critical, 0 severe, 0 moderate
The detailed report can be viewed online at http://localhost:8070/ui/links/application/test/report/9559...

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.618 s
[INFO] Finished at: 2014-02-20T10:46:42-08:00
[INFO] Final Memory: 25M/431M
[INFO] -----

Process finished with exit code 0
```

Figure 7.3: CLM for Maven Output in the Run Console in IntelliJ

7.3 NetBeans IDE

NetBeans IDE supports Maven projects natively and you can simply open a project in the IDE by choosing *Open Project* from the *File* menu and navigating to the directory that contains your project.

Once your project is opened, you can expand the *Project Files* section in the *Projects* window as displayed in Figure 7.4. Double-click on the `pom.xml` file and add the plugin configuration for Sonatype CLM for Maven from [Example Configuration of Sonatype CLM for Maven](#).

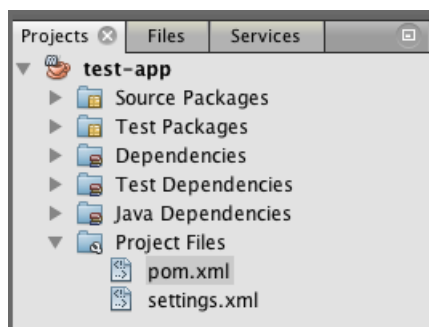


Figure 7.4: Project View with the `pom.xml` in NetBeans

If you right-click on the `pom.xml` file, you can choose *Run Maven* and *Goals*, to display the dialog displayed in Figure 7.5. Enter the configuration as displayed and don't forget to select *Remember as:* providing a name. This will allow you to simply start this defined configuration from the *Run Maven* context menu of the `pom.xml` file.

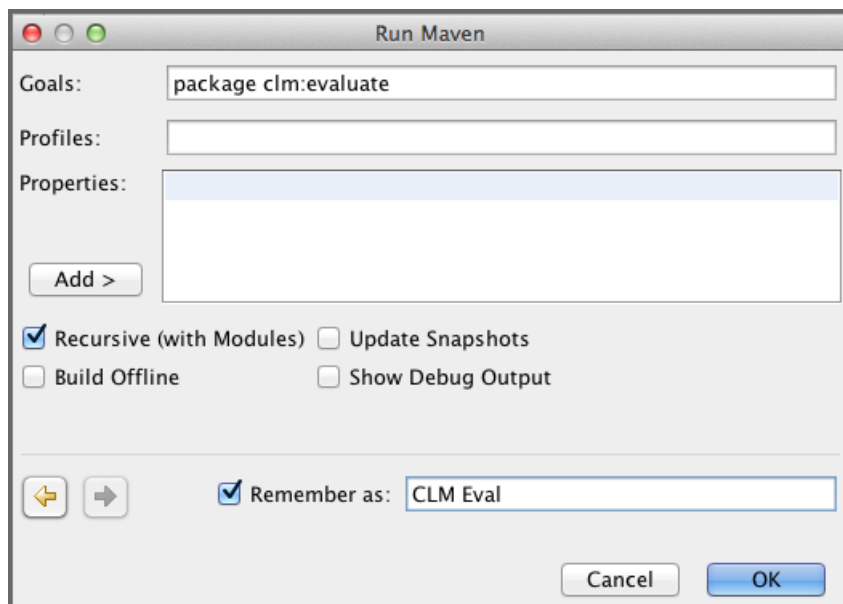


Figure 7.5: Maven Goal Setup for a CLM Evaluation in NetBeans

After pressing *OK* the defined Maven execution will start and display the output including any error messages and the link to the produced report in the Sonatype CLM server in the Output window displayed in Figure 7.6. Policy violations can be configured to result in a build failure.

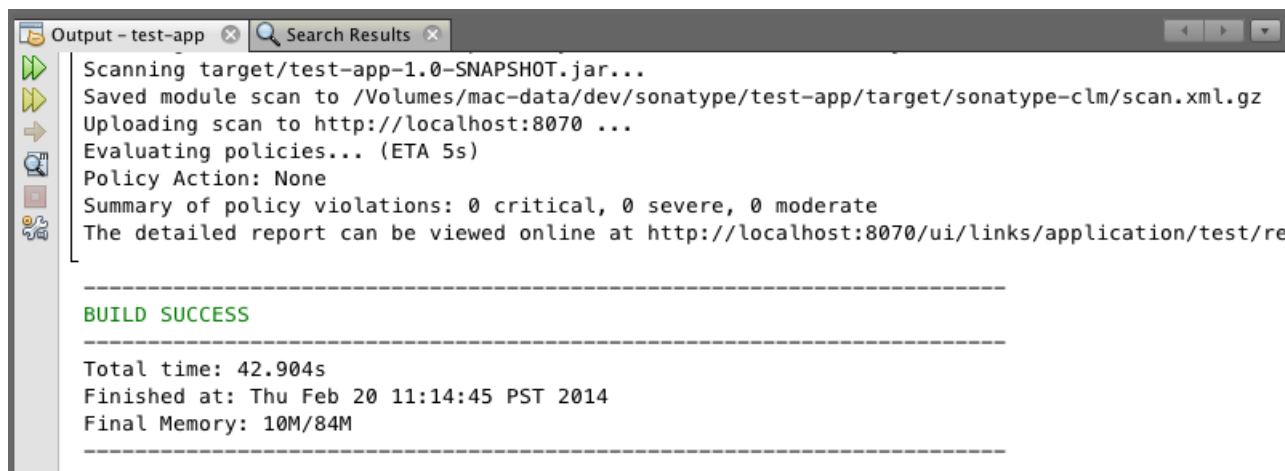


Figure 7.6: CLM for Maven Output in the Output Window in NetBeans

Chapter 8

Conclusion

As you likely discovered, the Sonatype CLM for Maven plugin not only provides direct command line Maven functionality, but can extend the existing features of tools that already have direct Sonatype CLM integration. For example did you look at our section on using the index goal to create a list of dependencies for your project?

When it comes down to it, Sonatype CLM for Maven allows you to ensure any Maven-based project meets your team's desire of reducing risk in association with open source component usage. Even if you are already using the Sonatype CLM functionality, exploring the capabilities of Sonatype CLM for Maven is still a good idea.

Take a look at a few of the key take aways to make sure you didn't miss anything:

- Creating a Component Index
- Evaluating a Maven Project from the Command Line
- Integrating with various IDEs (e.g. Netbeans, IntelliJ IDEA)