Step 7 - Sonatype CLM and Continuous Integration Server Usage (optional)

Contents

List of Figures

Return to the Nine Steps Main Page

Chapter 1

Introduction

At the core of open source governance with Sonatype CLM is the concept of enforcement points. An enforcement point represents a stage in the component and development lifecycle, for example, the CI exists in what we refer to as the *Build* CLM stage.

At each CLM stage, and in conjunction with policy, you have the opportunity to take specific actions. These can range from providing a warning, creating a failure, or sending out email communication. Of course, all of these are based on a component, or components, violating your policies.

This guide will walk you through installation, configuration, and basic usage for the Sonatype CLM for CI enforcement point. It is important to remember that policy is still managed via the Sonatype CLM Server, which is covered in the first six steps.

Note
Sonatype CLM for CI is an optional step to the Nine Steps for Open Source Governance. Depending on your particular purchase, you may not have access to this tool.

Chapter 2

Sonatype CLM for CI

2.1 Introduction

Eclipse Hudson and Jenkins are powerful and widely used open source continuous integration servers providing development teams with a reliable way to monitor changes in source control and trigger a variety of builds. They excel at integrating with almost every tool you can think of.

Historically the Hudson project and community split into two groups, with Jenkins as well as Hudson emerging as sibling products with a different focus going forward while sharing a common API for plugins. In general, with regard to the Sonatype CLM for CI functionality, the interaction will be near identical, with only a few differences, which are inherent to the CI, and not Sonatype CLM.

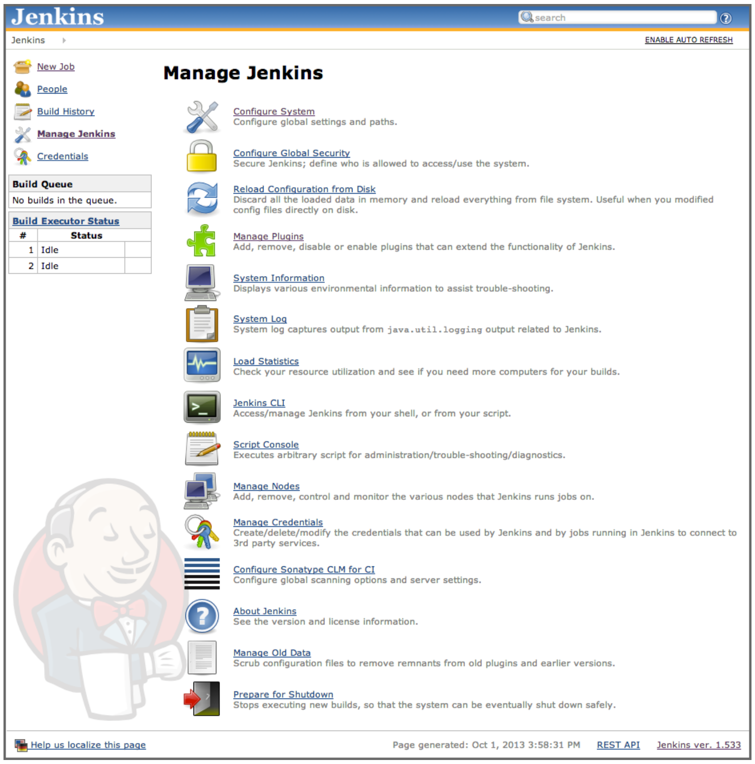
The Sonatype CLM for CI plugin scans the project workspace after a build for all supported component types, creates a summary file about all the components found and submits that to the Sonatype CLM service. The service uses that data to produce the analysis with the security and license information and send it back to the CI server. It will then use these results to render the analysis reports.

The file types supported for analysis are in tar/zip like format with the extensions tar, tar.bz2, tb2, tbz, tar.gz, tgz and zip or in Java archive formats of the type jar, ear, war, hpi, wsr, har, sar, rar, mar and nbm.

2.2 Installation

The Sonatype CLM for CI plugin is distributed as a Hudson plugin package (`.hpi` file) and is compatible with Jenkins and Hudson.

In order to install the plugin you have to log into Jenkins or Hudson as administrator and then select to *Manage Jenkins/Manage Hudson* to get to the global configuration menu displayed in Figure 2.1 in the Jenkins look. The Hudson look will be similar in content, yet different in colors and styling.



Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Manage Jenkins

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

System Information
Displays various environmental information to assist trouble-shooting.

System Log
System log captures output from `java.util.logging` output related to Jenkins.

Load Statistics
Check your resource utilization and see if you need more computers for your builds.

Jenkins CLI
Access/manage Jenkins from your shell, or from your script.

Script Console
Executes arbitrary script for administration/trouble-shooting/diagnostics.

Manage Nodes
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Manage Credentials
Create/delete/modify the credentials that can be used by Jenkins and by jobs running in Jenkins to connect to 3rd party services.

Configure Sonatype CLM for CI
Configure global scanning options and server settings.

About Jenkins
See the version and license information.

Manage Old Data
Scrub configuration files to remove remnants from old plugins and earlier versions.

Prepare for Shutdown
Stops executing new builds, so that the system can be eventually shut down safely.

Help us localize this page   Page generated: Oct 1, 2013 3:58:31 PM   REST API   Jenkins ver. 1.533

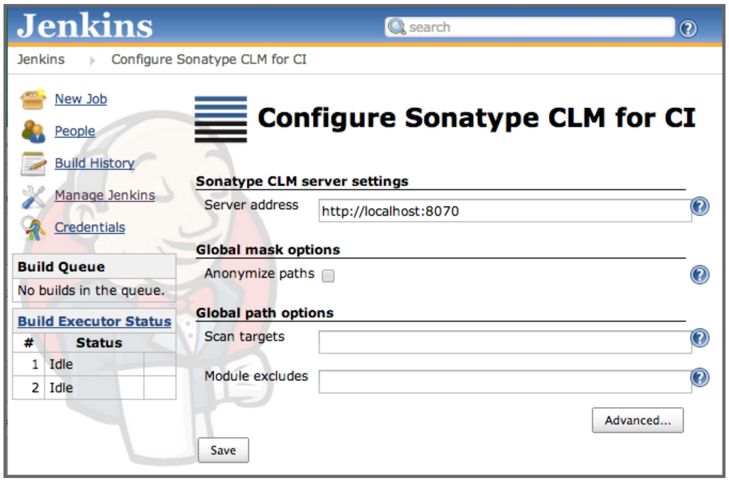
Figure 2.1: Jenkins Global Configuration Menu

From the displayed configuration menu, select *Manage Plugins* and in the plugin management section, choose the *Advanced* tab.

The advanced plugin management allows you to upload a plugin distribution file (`.hpi`) in the section entitled *Manual Plugin Installation* on Hudson and *Upload Plugin* on Jenkins. Click on *Choose File* and select the Sonatype CLM for CI plugin hpi file named `sonatype-clm-ci-x.y.z.hpi` with `x.y.z` representing a version number like `2.11.2` in the file selection dialog. Then press the *Upload* button. Once the plugin has been uploaded to the server, you need to restart your continuous integration server.

2.3 Global Configuration

After a successful installation of the Sonatype CLM for CI plugin, the global Jenkins/Hudson configuration menu, displayed in Figure 2.1 includes a separate item for Sonatype CLM with the title *Configure Sonatype CLM for CI* . Click the link to get to the global configuration displayed in Figure 2.2.



Configure Sonatype CLM for CI

Sonatype CLM server settings

Server address: http://localhost:8070

Global mask options

Anonymize paths ☐

Global path options

Scan targets

Module excludes

#	Status
1	Idle
2	Idle

Figure 2.2: Global Configuration of Sonatype CLM for CI in Jenkins

The global configuration for Sonatype CLM for CI is used as the default configuration for all invocations of the plugin. Specific parameters supplied for individual jobs are appended to the global configuration. You can configure the following settings:

Sonatype CLM server settings

Server address
The address for the Sonatype CLM server as it can be reached from the Jenkins/Hudson server. The address should be the same one a user is using to access the Sonatype CLM

server interface. A suitable URL for a default install on your local computer would be `http://localhost:8070`. If your Sonatype CLM server is behind a proxy server for serving HTTPS or other reasons, you have to use the public URL as it is reachable from the continuous integration server. Only the master Jenkins/Hudson server connects to the CLM server and you therefore only need ensure connectivity in terms of open firewall ports and proxy server settings between the master CI server and the CLM server. This configuration parameter is the only required setting.

Global mask options

Anonymize paths

Enabling this feature will anonymize all paths before data is sent to the Sonatype CLM server. Ultimately, this prevents the CLM report from reporting the locations/occurrences of components. Our recommendation is to leave this disabled, unless you are worried about Sonatype knowing about the file names of your components.

Global path options

Scan targets

The scan targets setting allows you to control which files should be examined. The configuration uses an Apache Ant styled pattern, is relative to each project's workspace root directory, and has a useful default setting that includes all `jar`, `war`, `ear`, `zip` and `tar.gz` files. The default value is therefore

```
**/*.jar, **/*.war, **/*.ear, **/*.zip, **/*.tar.gz
```

Note

This default only applies if and only if neither global nor job config specify scan targets.

Module excludes

As part of CLM, Sonatype has included a CLM Maven Plugin. Use of the CLM Maven plugin in the build process will result in the creation of module information files. If desired, you can exclude some of the modules from being scanned. The default location where the modules are stored is `${project.build.directory}/sonatype-clm/module.xml`.

To exclude a module, use a comma-separated list of Apache Ant styled patterns relative to the workspace root that denote the module information files (`**/sonatype-clm/module.xml`) to be ignored, e.g.

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

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

Tip

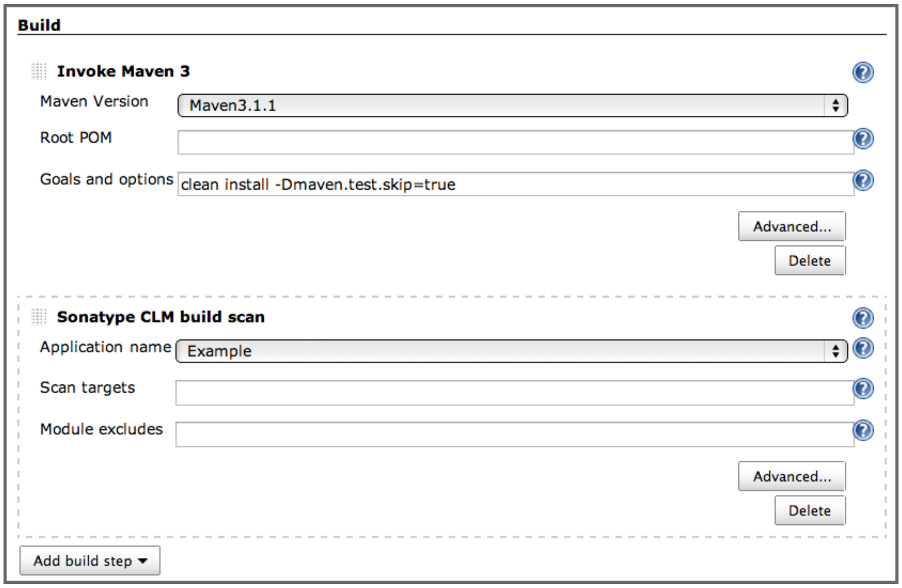
While the CLM Maven Plugin produces these files for all modules in a Maven build, the excludes can be used to ignore certain modules from analysis. For more on the CLM Maven Plugin, see Section 4.1.

Advanced options
A number of additional parameters can be supplied to the plugin using this input field. Typically these parameters will be determined by Sonatype support.

2.4 Job Configuration

After a completed installation (see Section 2.2) and global configuration (see Section 2.3) of Sonatype CLM for CI, you are ready to configure an invocation as part of a specific job.

Depending on your job type it will be available as pre and/or post-build step as well as a invocation as a main build step. The typical invocation would be as main build step, after the package that should be examined has been created. An example configuration from Jenkins is displayed in Figure 2.3. Alternatively a post-build step for example as displayed in Figure 2.4 can be used as well. A pre-build step or a main build step executed before your main build invocation step could be used to examine components existing in the workspace or being placed into the workspace by an earlier build step.


Build

Invoke Maven 3

Maven Version: Maven3.1.1

Root POM:

Goals and options: clean install -Dmaven.test.skip=true

Sonatype CLM build scan

Application name: Example

Scan targets:

Module excludes:

Figure 2.3: Sonatype CLM Build Scan Configuration for a Build Step

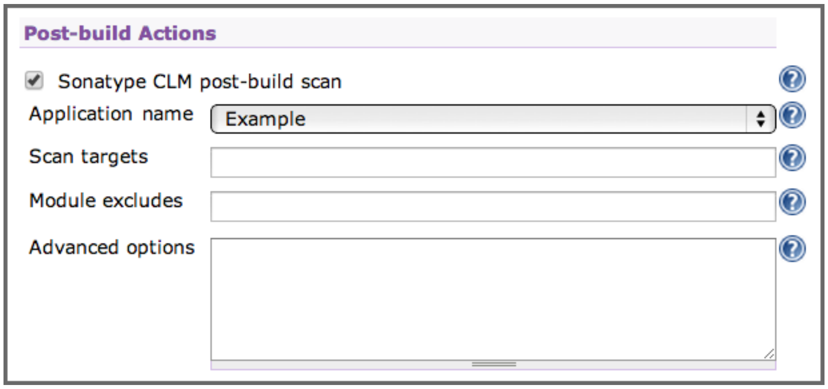
The configuration options for Sonatype CLM for CI invocations mimic the parameters from the global configuration described in Section 2.3 and are appended to the global parameters. The configuration parameters are:

Application name
The drop down for application name should be populated with the name of all applications configured in your Sonatype CLM server and allows you to select the desired application scanning configuration. The policies associated to the application will be used for the analysis of this build job output.

Scan targets
The scan targets setting allows you to control which files should be examined with an Apache Ant styled pattern. The pattern is relative to the project workspace root directory and inherits the global configuration.

Module excludes
You can exclude modules from being scanned with module information files configured in this setting. The default value is inherited from the global configuration.

Advanced options
A number of additional parameters can be supplied to the plugin using this input field. Typically these parameters will be recommended to you by the Sonatype support team.


Post-build Actions

✓ Sonatype CLM post-build scan	
Application name	Example
Scan targets	
Module excludes	
Advanced options	

Figure 2.4: Post-build Action Configuration as Example for a Sonatype CLM for CI Configuration

2.5 Inspecting Results

Once a specific build has successfully completed, Sonatype CLM for CI provides a link to the application composition report in the job list in the *Policy Violations* column as well as the project specific overview page. Clicking on the link *Application Composition Report*, will direct you to the display of the report within the Sonatype CLM Server. The three boxes (red, orange, and yellow), below the link give you counts for policy violation counts for the different serverities as a quick indication of the analysis results.

In addition to the link to the report, the left hand menu for the job includes *Application Management*. Clicking on the link will take you directly to the specific application on the Sonatype CLM Server. In Figure 2.5 you can see both the link to the report, and the link to Application management.

Note
Accessing this information may require a login. Also, if you are using a version of the Sonatype CLM for CI plugin prior to version 2.11, and Sonatype CLM Server 1.7, a message will display indicating your report has been moved. Following this link will take you to the report on the Sonatype CLM Server.

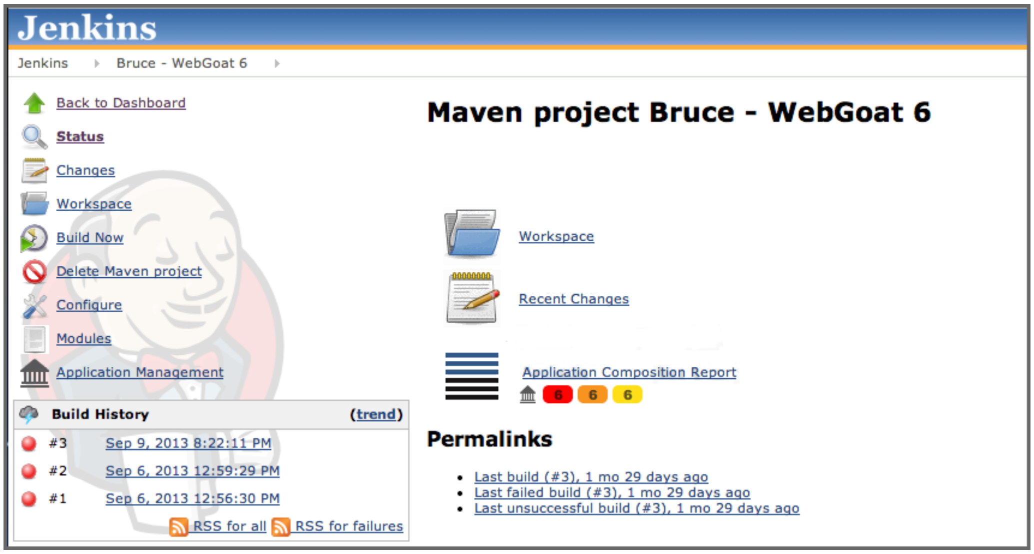
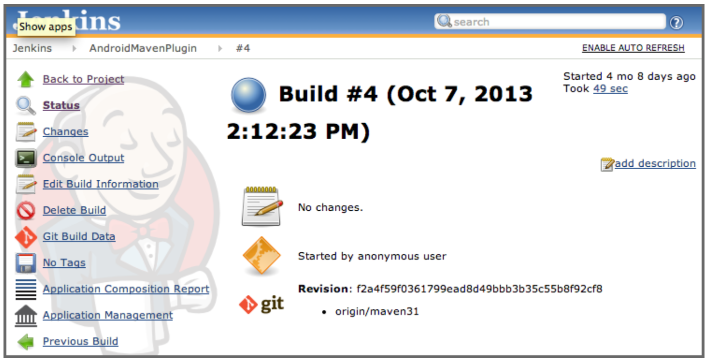

Jenkins
Jenkins › Bruce - WebGoat 6 ›
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Maven project
Configure
Modules
Application Management
Build History (trend)
#3 Sep 9, 2013 8:22:11 PM
#2 Sep 6, 2013 12:59:29 PM
#1 Sep 6, 2013 12:56:30 PM
RSS for all RSS for failures
Maven project Bruce - WebGoat 6
Workspace
Recent Changes
Application Composition Report
6 6 6
Permalinks
Last build (#3), 1 mo 29 days ago
Last failed build (#3), 1 mo 29 days ago
Last unsuccessful build (#3), 1 mo 29 days ago

Figure 2.5: Job Overview Page with Links to the Application Composition Report and Application Management

If you are looking for previous report results, simply navigate to a specific build in the *Build History*. If you have previously scanned the application during that specific build, you will see a new item in the left menu, *Application Composition Report*. As with the report link above, you will be taken to the Sonatype CLM Server to review the results. An example is show in Figure 2.6 below.



Build #4 (Oct 7, 2013 2:12:23 PM)

Started 4 mo 8 days ago
Took 49 sec

No changes.

Started by anonymous user

Revision: f2a4f59f0361799ead8d49bbb3b35c55b8f92cf8

- origin/maven31

Figure 2.6: Left Menu with Link to the Application Composition Report

Chapter 3

Sonatype CLM Command Line Scanner

3.1 Introduction

While the best integration between Sonatype CLM and a CI is through the fully supported add ons (e.g. Hudson/Jenkins), any CI server can easily be connected to your Sonatype CLM policies simply by using the Stand-alone, or CLI, version of the Sonatype CLM Scanner. In this section, we will walk you through the basic setup, and provide a sample syntax that can be used in most scenarios. In general, before you go this route, there are a few things you should make sure you have, including:

- A general familiarity with the CLI (you don't need to be an expert, but basic knowledge helps).
- Familiarity with the way your particular CI utilizes builds steps to launch a simple script.
- Installed and setup the Sonatype CLM Server
- Setup an organization and application
- Created or imported a policy for your application, or the organization that contains your application.

3.2 Downloading the Scanner

To use the stand-alone, Sonatype CLM scanner, first download the jar file named similar to `sonatype-clm-scanner-` from the Sonatype Support website and place the file in its own directory.

Note
Be sure to remember where you placed the scanner. As a recommendation, it's best to have the scanner in it's own directory, and not shared with the Sonatype CLM Server.

3.3 Locating Your Application Identifier

Again, before we scan, let's make sure we have everything you need. Make sure you have:

- Created an organization
- Created an application
- Imported or created a policy

Once those are done, make sure you know the application identifier, which can be found by:

1. Log into your Sonatype CLM server with a user account with at least a developer role for the application you plan on scanning.
2. Choose *Management* from the global navigation drop down menu.
3. Click on *Applications*, and then click the application. You should see a screen similar to Figure 3.1
4. Locate the text underneath the application name. The application identifier is the text between *Application* and *in [organization name]*

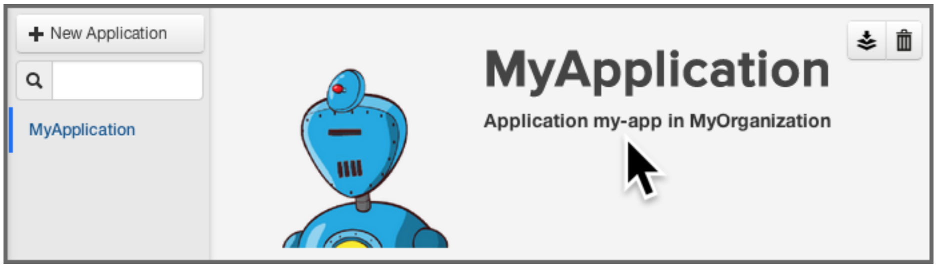


Figure 3.1: Application Overview and Application Identifier

3.4 Setting Up the Scanner in Your CI

We won't be covering a specific CI here, but in general, all you need to identify (in your CI), is the location for adding a build step that includes processing a simple shell script during the building of your application.

Once you are there, make sure your script calls the CLI scanner using the following syntax:

```
java -jar [ScannerJar] -i [AppID] -e [IgnoreSystemErrors] -w [ ↩
    FailOnPolicyWarning] -s [ServerURL] [Target]
```

Each of the areas in the syntax above have been described in more detail below.

ScannerJar
the path to the Sonatype CLM Scanner jar file e.g. `./sonatype-clm-scanner-1.7.0-02.jar`

AppId
the application identifier determined in Section 3.3

IgnoreSystemErrors
an optional feature that will allow you to continue a build even if the CLM scanner encounters an error (e.g. the CLM server can't be contacted).

FailOnPolicyWarning
an optional feature that allows you to fail a build if a policy triggers a warning alert. (Policy error alerts always fail the build.)

ServerUrl
the URL of your Sonatype CLM Server e.g. `http://localhost:8070`

`Target`
the path to a specific application archive file or a directory containing such archives. Archive files in a number of formats including `jar`, `war`, `ear`, `tar`, `tar.gz`, `zip` and many others are supported for scans.

Tip
In addition to the functionality described here, the stand-alone scanner, can be used to scan applications directly from the CLI. To access help content for the scanner run scanner without supplying parameters: java -jar ./sonatype-clm-scanner.jar

Given a typical setup, your syntax, including all available options will likely look similar to this:

```
java -jar /scanner/sonatype-clm-scanner.jar -i tester123 -s http:// ↩
    localhost:8070 ./target/sample-app.war
```

Now, when your application is built, the build step you have added will call the stand-alone Sonatype CLM scanner, scan your application, and upload results of the latest scan to Sonatype CLM Server. By default this will be placed below the build column in the Reports and Application area on the Sonatype CLM Server, for your application.

Note
We advise you to use a separate application identifier for each of your unique applications. Using the same application identifier will result in report results being overwritten each time an application is built. While this is always the case, matching the latest scan to the right application can prove difficult.

3.5 Summary

We breezed through this setup pretty quickly, but it really is that simple. That's because most of the heavy lifting (creating applications and organizations, as well as establishing policy) should have been done prior to this step. If you've done that, getting the CLI scanner integrated is as easy as just copying our sample syntax and copying in your own, specific parameters.

It's important to remember though, this process only covers a single application. For each additional application you specify the specific application identifier.

Chapter 4

Sonatype CLM Maven Plugin

4.1 CLM Maven Plugin Introduction

A Sonatype CLM evaluation of a Maven based software project can be assisted by the Sonatype CLM Maven plugin. It can take advantage of the dependency information contained in the project's `pom.xml` files and the information about transitive dependencies available to Maven. It can be run on a command line interface and can therefore be executed on any continuous integration server.

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. However you can also analyze a all modules of a project. This will largely depend on what your CLM policy is enforcing and what you want to validate.

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

The `attach` goal aids your integration with Sonatype Nexus CLM Edition and the release process using the staging tools of Nexus.

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

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:2.1.1:help
```

The following sections detail a these goals and their usage.

4.2 Creating a Component Index for Sonatype CLM for CI

The `index` goal of the CLM Maven plugin allows you to identify component dependencies and makes this information available to the Sonatype CLM for CI plugin. 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>
```

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 the CLM Maven plugin is able to create a complete dependency list rather than relying on binary build artifacts.

Only dependencies in the `compile` or `runtime` scopes will be considered, since this reflects what other Maven packaging plugins typically include. Dependencies with the scopes `test`, `provided` and `system` will not be considered.

4.3 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 scan.

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>
```

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 an 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 the CLM Maven plugin is able to create a complete dependency list rather than relying on binary build artifacts.

4.4 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 `procure`, `develop`, `build`, `stage-release`, `release` and `operate` with a default value of `build`.

An example invocation is:

```
mvn com.sonatype.clm:clm-maven-plugin:evaluate
-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>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

The CLM Maven plugin 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 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 multimodule 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.

4.5 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
```

4.6 Skipping CLM Maven Plugin 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 5

Summary

Alright, Sonatype CLM for CI is now installed and configured, and will be included as part of your open source governance initiative. If you have purchased a license for the IDE and/or Nexus Pro - CLM Edition, you can move on to the next steps:

- Sonatype CLM for IDE Installation and Configuration
- Sonatype Nexus Pro - CLM Edition Installation and Configuration