




BLACKDUCK

Installing, Configuring, and
Using the Hub Plugin for
Jenkins

Version 2.1.0



This edition of the *Installing, Configuring, and Using the Hub Plugin for Jenkins* refers to version 2.1.0 of the Black Duck Hub Plugin for Jenkins.

This document created or updated on Thursday, February 02, 2017.

Please send your comments and suggestions to:

Black Duck Software, Incorporated
800 District Avenue
Suite 221
Burlington, MA 01803 USA.

Copyright © 2017 by **Black Duck Software, Inc.**

All rights reserved. All use of this documentation is subject to the license agreement between Black Duck Software, Inc. and the licensee. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the prior written permission of Black Duck Software, Inc.

Black Duck, Know Your Code, and the Black Duck logo are registered trademarks of Black Duck Software, Inc. in the United States and other jurisdictions. Black Duck Code Center, Black Duck Code Sight, Black Duck Export, Black Duck Hub, Black Duck Protex , and Black Duck Suite are trademarks of Black Duck Software, Inc. All other trademarks or registered trademarks are the sole property of their respective owners.

Chapter 1: Hub Jenkins Plugin Overview	5
1.1 Supported Archive Types	5
1.2 Hub Jenkins Plugin Requirements	6
1.3 Updating the Hub Jenkins plugin	6
Chapter 2: Installation Overview	8
2.1 Installation Prerequisites	8
2.2 Downloading and Installing the Hub Jenkins Plugin	8
Chapter 3: Configuring the Hub Jenkins Plugin	9
3.1 Configuring a Proxy	11
3.2 Configuring Hub Jenkins Logging	11
3.3 Server Response at the Global Configuration Level	12
3.4 Configuring Connection Timeout	12
3.5 Performing a Workspace Check	13
Chapter 4: Using the Hub Jenkins Plugin	14
4.1 Code Locations	14
4.2 Cleaning Up Logs on Successful Scans	15
4.3 Directory Exclusion Pattern	16
4.4 Dry Run Option	17
4.5 Configuring Hub Risk Reports	17
4.6 Generating Reports	18
4.7 Hub Failure Conditions	19
4.8 Setting Configurable Build State	21
Chapter 5: Hub Jenkins Best Practices	22
5.1 Permissions	22
5.2 Post Build Action Blocking	22
5.3 Sharing Workspace with a Downstream Job	22
5.4 Parameterized Job Configuration	23
5.5 Downstream Job Configuration	25
5.6 Multijob Configuration	26
5.7 Best Practices: Relative Paths	28
Chapter 6: Troubleshooting the Hub Jenkins Plugin	30
Chapter 7: Hub Jenkins Plugin Release Notes	31
7.1 Hub Jenkins Known Issues	33

Chapter 8: Black Duck Support	34
8.1 Training	34
8.2 Services	35

Chapter 1: Hub Jenkins Plugin Overview

Black Duck Hub is a new risk management tool designed to help you manage the logistics of using open source software in your organization.

The Black Duck Hub Scanner is the software component scanning functionality in Black Duck Hub that provides an automated way to determine the set of open source software (OSS) components that make up a software archive. The Hub Scanner is designed to help organizations manage their use of open source binaries by identifying and cataloging OSS components to provide additional metadata such as license, vulnerability, and OSS project health for those components.

Jenkins is an open source continuous integration tool that monitors executions of repeated jobs, such as building a software project or cron jobs. Jenkins focuses on building and testing software projects continuously and monitoring execution of externally run jobs.

As a Hub and Jenkins user, the Hub Jenkins plugin enables you to:

- Run a component scan in a Jenkins job:
 - Scan multiple targets within the job workspace.
 - Define the component scan command line interface (CLI) as a tool.
 - Create projects and releases in Black Duck Hub through the Jenkins job.
- After a scan is complete, the results are available on the Hub server.

Using the Hub Jenkins Plugin together with the Hub Scanner lets you use Jenkins to automatically create Hub projects from your Jenkins projects.

1.1 Supported Archive Types

The Hub Scanner can scan archive files as well as a directory of files. The following archive file types can be processed by the Hub Scanner:

The Black Duck Component Scanning can extract the following archive types:

- AR
- ARJ
- CPIO
- DUMP
- TAR
- RPM
- ZIP
- 7z

Archives may optionally be compressed using any of the following compression algorithms:

- Bzip2
- Gzip
- Pack200
- XZ
- LZMA
- Snappy
- Z (compress)
- DEFLATE

Note: If you attempt to scan an individual archive file that is not a supported type, the Hub Scanner finds no matches.

1.2 Hub Jenkins Plugin Requirements

Software Requirements

The installation instructions in this document assume that you have the following installed and configured on your system:

- Black Duck Hub 3.5 or higher
- Jenkins 1.580.3 or higher (we recommend the latest LTS release)
- Java SE 7

Note: The Hub Scanner CLI client requires Java Runtime Environment (JRE) version 1.7.0_40 or later to be installed on the computer where it is run. For Hub versions 3.0 and higher, this is not required.

- Maven 3
- Microsoft Build version 11 or 12. Note that Visual Studio 2012 installs Microsoft Build 11, and Visual Studio 2013 installs Microsoft Build 12.
- Microsoft .NET Framework version 4.5

The Jenkins plugin is supported on the same operating systems and browsers as Black Duck Hub.

Network Requirements

The Hub Jenkins plugin requires internet connectivity. The machine that hosts your Jenkins server must be able to connect to the Hub server.

1.3 Updating the Hub Jenkins plugin

You can update the Hub Jenkins plugin as new versions are released.

✳ **To update the Hub Jenkins plugin:**

1. Navigate to **Manage Jenkins > Manage Plugins**.
2. Click **Updates**.
3. Select **Black Duck Hub Plug-In for Jenkins**.
 - a. If there are updates for **Black Duck Hub Plug-In for Jenkins**, the updates display in the list.
 - b. Alternatively, you can force Jenkins to check for plugin updates by clicking **Check now**.
4. If there are updates, select the one you want, and click **Download now and install after restart**.
5. You then have the option of restarting and performing the install either now or later.

Note: You can find Jenkins documentation at:
<https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins>

2.1 Installation Prerequisites

Before you install the Hub Jenkins plugin, ensure that:

- Your Jenkins instance is up-to-date and fully patched.
- You know the host name and port for the Hub server.
- You have a user account with administrator privileges on the Hub system that you can use for the integration.
- You have connectivity to the internet. The machine that hosts your Jenkins server must be able to connect to the Hub server.

2.2 Downloading and Installing the Hub Jenkins Plugin

* To download the Hub Jenkins plugin:

1. Navigate to <https://github.com/blackducksoftware/jenkins-hub/releases>.
2. Download and save the `.hpi` file to a temporary location.

* To install the Hub Jenkins plugin:

1. Log in to Jenkins as administrator.
2. Navigate to **Manage Jenkins > Manage Plugins**.
3. Click the **Advanced** tab.
4. In the **Upload Plugin** section, click **Browse** and navigate to the location of the plugin `.hpi` file and select it.
5. Click **Upload**.
6. Restart the Jenkins instance.

Note: As of Hub Jenkins version 2.0.0, the plugin is in the Jenkins marketplace. The version 2.0.0 installation may result in two installations of the Hub Jenkins plugin within your Jenkins instance. The old version (pre-2.0.0) must be uninstalled.

Chapter 3: Configuring the Hub Jenkins Plugin

* To configure the Hub Jenkins plugin:

1. In Jenkins, click **Manage Jenkins** > **Configure System**.
2. In the Black Duck Hub section:
 - a. In the **Server URL** field, enter the URL of the Hub server, including the port number; for example:
`http://hub.blackducksoftware.com:8080`
 - b. In the **Credentials** field, select your credentials for the Hub server from the menu. If your credentials are not in the list, you can add them by clicking the **Add** button.
 - c. Test the connection to the server with the selected credentials by clicking the **Test Connection** button.

The screenshot shows the 'Black Duck Hub' configuration section in Jenkins. It contains three main fields: 'Server URL' with the value 'http://eng-hub-valid01.dc1.lan', 'Credentials' with a dropdown menu showing 'sysadmin/***** (integration hub)' and an 'Add' button, and 'Connection Timeout' with the value '120' and the unit 'in seconds'. A 'Test Connection' button is located at the bottom right of the form.

3. Click **Save**.
4. Go back to the Jenkins dashboard.
5. Click a job.
6. Click **Configure**.
 - a. If multiple JDKs are defined, select the one to use from the **JDK** menu.
 - b. In the **Post-build Actions** section, click **Add post-build action**.
 - c. Select *Black Duck Hub Integration* from the list.

The screenshot shows the 'Post-build Actions' section of a Jenkins job configuration. Under the 'Black Duck Hub Integration' section, the following fields are visible:

- Project Name:** Hub-Jenkins 2.1.0 on MASTER. Below it, a message states: 'This Project exists on the Hub Server : http://qa-hub07.dc1.lan:8080'.
- Project Version:** Running on Master. Below it, a message states: 'This Version exists in the Project : Hub-Jenkins 2.1.0 on MASTER'.
- ☐ Generate Black Duck Risk Report
- Maximum time to wait for BOM update (in minutes):** 5
- Scan Memory Allocation:** 4096
- Code Location Name:** (empty field)
- ☐ Dry Run
- ☐ Cleanup logs on successful scan
- Scan Target:** (empty field). Below it, a yellow warning message says: 'The scan target is empty, the entire workspace will be scanned'.
- Buttons: 'Delete Scan target' (red), 'Add another Scan target' (grey).
- Directory Exclusion Pattern:** (empty field)
- Buttons: 'Save' (blue), 'Apply' (grey).

- d. In the **Black Duck Hub Integration** section, enter the name of your project in the **Project Name** field. When you start typing, the plugin makes suggestions based on existing projects in the Hub. Leave the field empty if you do not want the scan automatically mapped to a particular project.
- e. In the **Project Version** field, enter the name of the project version. When you start typing, the plugin makes suggestions, based on existing releases in the Hub. Leave the field empty if you do not want the scan automatically mapped to a particular release. The **Project Name** and **Scan Target** fields accept variables as part of the input. Variables can be defined as: \$, { . . }, or \$ For example, \${JOB_NAME} and \$JOB_NAME are both accepted as variables and are resolved during the build.

Note: In Hub Jenkins if you are running builds on a Windows slave, you cannot configure a project name and version using non-Windows encoding characters; for example, Chinese characters. Therefore, the project name and version cannot contain Unicode characters. This is not an issue in Linux or Mac operating systems.

- f. You can display the Black Duck Hub risk report within Jenkins.
- g. Change the **Scan Memory Allocation** value to be at least 4000 MB or greater. This setting is configured in MB, but the value should be more than 4 GB; in other words, greater than 4000 MB.
- h. In the **Scan Target** field, enter the path to the file you want to scan.
- i. To define additional targets, click **Add another Scan target**.

Note: If you leave the target empty, the entire workspace is scanned.

7. Click **Save**.

Note: You can no longer configure the CLI, because Black Duck defines the tool and automatically installs the CLI.

Caution: In previous versions, you were required to override the CLI tools per slave; this is no longer required. Doing so is still possible but not recommended; override the CLI at your own risk.

3.1 Configuring a Proxy

As of Hub Jenkins versions 2.1.0 and higher, the proxy functionality is:

- Basic with and without authentication: works for http and https Hub servers.
- Digest authenticated proxy is no longer supported.

* To configure the Hub Jenkins plugin to use a proxy:

1. Log in to Hub Jenkins as administrator.
2. Go to **Manage Jenkins > Manage Plugins**.
3. Click the **Advanced** tab.
4. Enter information for the proxy:
 - **Server**
 - **Port**
 - **User name**
 - **Password**
 - **No Proxy Host**
5. To verify your proxy configuration, click **Advanced...**
6. Type your test URL in the **Test URL** field; then click **Validate Proxy**.
7. Click **Submit**.

3.2 Configuring Hub Jenkins Logging

Beginning with Hub Jenkins version 1.5.1, you can configure your logging level.

In configuring the logging level, Hub Jenkins examines the `HUB_LOG_LEVEL` environment variable.

- If this variable is set with a valid value, this value defines the logging level.
- If this variable is not set, then the default `INFO` logging level is used.

Setting the logging level

There are two ways of setting the logging environment variable.

- In the global configuration, located in **Manage Jenkins > Configure system**.
- At the job level.

Logging level values

You can specify the following log levels.

- OFF
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Note: The *Env Inject* plugin is required for the job level logging configuration.

3.3 Server Response at the Global Configuration Level

You can see the Hub server response in the Global configuration screen when you input the Hub URL used within your Jenkins system. You can:

- View the Hub server status by the Hub server address within the Global configuration screen.

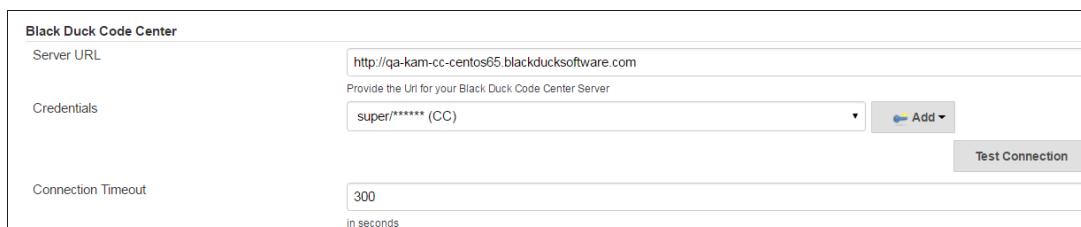
In the event a server goes down after a response has been registered while defining the Hub URL, you can go back to the global configuration screen to check its status in real-time.

3.4 Configuring Connection Timeout

You can set the server connection timeout values in the Hub Jenkins plugin.

* To set the connection timeout:

1. Log in to Hub Jenkins as administrator.
2. Go to **Manage Jenkins > Configure System > Global Configuration > Black Duck Hub**.
3. Click **Advanced**, and complete the following.



The screenshot shows the 'Black Duck Code Center' configuration section in Jenkins. It contains three main fields: 'Server URL' with the value 'http://qa-kam-cc-centos65.blackducksoftware.com', 'Credentials' with a dropdown menu showing 'super/***** (CC)' and an 'Add' button, and 'Connection Timeout' with a value of '300' and the unit 'in seconds'. A 'Test Connection' button is located to the right of the 'Credentials' field.

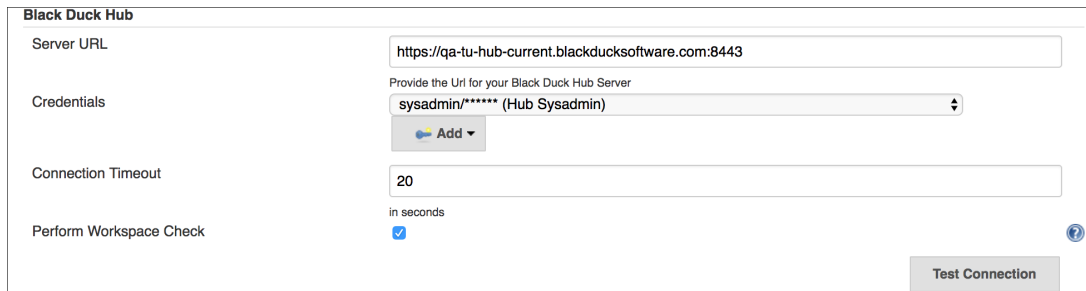
4. In the **Connection Timeout** field, type the number of seconds to wait before the server connection times out. The default value is 300 seconds.

3.5 Performing a Workspace Check

After configuring your Hub Jenkins plugin, you can select the workspaces to scan.

* To perform a workspace check:

1. Log in to Hub Jenkins as administrator.
2. Go to **Manage Jenkins > Configure System > Global Configuration > Black Duck Hub**.
3. Click **Advanced**, and complete the following.



The screenshot shows the 'Black Duck Hub' configuration page in Jenkins. It includes the following fields and options:

- Server URL:** A text input field containing 'https://qa-tu-hub-current.blackducksoftware.com:8443'.
- Credentials:** A dropdown menu showing 'sysadmin/***** (Hub Sysadmin)' with an 'Add' button below it.
- Connection Timeout:** A text input field containing '20', with a note 'in seconds' below it.
- Perform Workspace Check:** A checkbox that is checked.
- Test Connection:** A button located at the bottom right of the configuration area.

4. **Perform Workspace Check:** When checked, this enables you to point to any target location on your Jenkins node for scanning. It performs a workspace check during builds, ensuring that all targets are within the workspace, and allows you to scan targets outside the workspace.

Chapter 4: Using the Hub Jenkins Plugin

You can use the Hub Jenkins plugin to scan jobs, view the resulting message output, view scan results, and view scan logs.

* To use the Jenkins plugin to scan a job:

1. In Jenkins, go to the configured Job.
2. Click the **Build Now** link.

You can view the messages output from the **Jenkins Console Output** screen. In Jenkins, click the build, then click the **Console Output** link.

You can also view the logs from the Hub Scanner in the following directory:

```
{WORKSPACE}/HubScanLogs/{BUILD_NUMBER}/log
```

You can view the results of the Hub Scanner on the Hub server. In Hub, click the expanding menu icon, then click **Component Scans**.

- To map a scan to a project, click **Map to Project**.
- To view the BOM, click **Bill of Materials**.
- To view the mapped project, click the project name.
- To view the mapped release, click the release name.

Alternatively, you can also view the logs from the Black Duck scans in the directory `{SCAN_HOME}/lib/log`.

Note: To maximize scanning capability, the Hub command-line interface (CLI) as invoked by the Jenkins Hub plugin may require third-party native installs, such as NPM (Node Package Manager) for Node.js applications, to be available on the Jenkins server or remote slave nodes. For more information, refer to the *Best Practices* topic in the Black Duck Hub online Help.

4.1 Code Locations

Hub Jenkins plugin versions 2.1.0 and higher features a new **Code Location** field on the configuration page. Using the code location functionality can make your continuous integration build process smoother and easier, and allows for multiple users scanning the same code base. Using the **Code Location** field is optional.

Black Duck Hub Integration

Project Name: JR Test
This Project exists on the Hub Server : http://int-hub01.dc1.lan:8080

Project Version: 2.0
This Version exists in the Project : JR Test

☐ Generate Black Duck Risk Report

Maximum time to wait for BOM update (in minutes): 5

Scan Memory Allocation: 4096

Code Location Name:
This will change the name of the Code Location that is created by this scan.
An example of a consistent Code Location across nodes and builds would be `${JENKINS_URL}-${JOB_NAME}`
(from [Black Duck Hub Plugin for Jenkins](#))

☐ Dry Run
☐ Cleanup logs on successful scan

The **Code Location** field is hidden by default; click **Advanced** to display this field. The command line interface uses the "- -" (dash dash) name option. The **Code Location** field automatically adds the "- -" value. The **Code Location** field:

- Allows you to supply the name of your code location.
- Creates the command for you.
- Automatically adds it to the build action.

Continuous integration build processes have slaves or agents. The **Code Location** field automatically creates a local workspace on the machine issuing the commands. However, the target name is different, and is *host name* followed by *path*.

If two users are scanning the same code on continuous integration systems, the code location functionality automatically updates the Bill of Materials when more than one instance of the scan occurs, as long as both code locations match.

The **Code Location** field accepts Jenkins variable syntax. One example of a consistent code location across nodes and builds is `${JENKINS_URL}-${JOB_NAME}`. If you are using Jenkins continuous integration, you must use Jenkins variable syntax.

4.2 Cleaning Up Logs on Successful Scans

Hub Jenkins plugin versions 2.1.0 and higher provide functionality to automatically remove log files for successful scans. This functionality can save substantial hard drive space, and improve performance.

* To automatically delete log files on successful scans:

1. On the configuration page, click **Advanced**.
2. In the advanced options, click the checkbox for **Cleanup logs on successful scan**.



Code Location Name

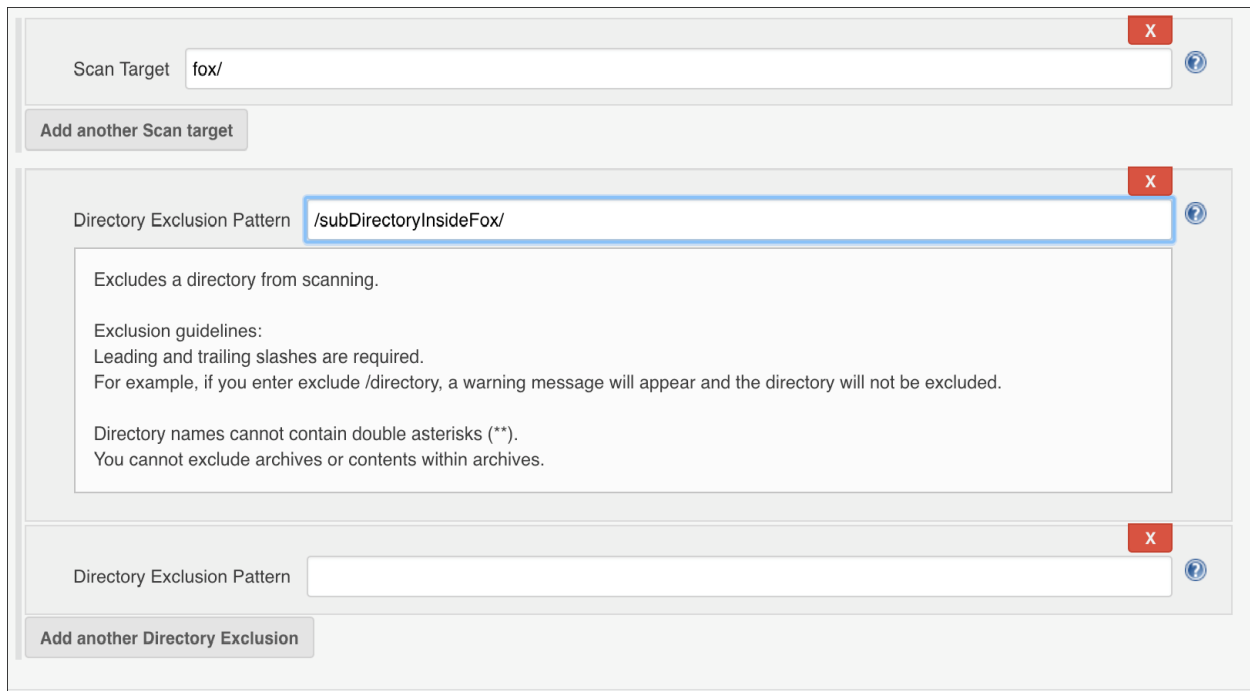
☐ Dry Run

☐ Cleanup logs on successful scan

If the scan has no errors, then no logs are generated when this field is selected. Note that the default option is unchecked.

4.3 Directory Exclusion Pattern

Hub Jenkins plugin versions 2.1.0 and higher feature Directory Exclusion Pattern functionality. This is found in the **Directory Exclusion Pattern** field on the **Scan Target** page. This enables you to specify sub-directories to exclude from scans.



Scan Target

Add another Scan target

Directory Exclusion Pattern

Excludes a directory from scanning.

Exclusion guidelines:
Leading and trailing slashes are required.
For example, if you enter exclude /directory, a warning message will appear and the directory will not be excluded.

Directory names cannot contain double asterisks (**).
You cannot exclude archives or contents within archives.

Directory Exclusion Pattern

Add another Directory Exclusion

Directory Exclusion Pattern syntax rules:

The Directory Exclusion Pattern functionality follows the GIT *ignore* pattern. The syntax is:

`/*name_of_sub-directory_to_exclude/`

- Leading and trailing forward slashes are required; in other words, the exclusion pattern must start with `/` and end with `/`.
- Directory names cannot contain double asterisks (**).

Using Directory Exclusion Pattern ignores all contents of the specified sub-directory during scans. Additionally, you can specify full paths; for example, the command `/*Parent/Child1/Child2/` excludes the *Child2* sub-directory from the scan. Directory Exclusion Pattern also performs validation.

To add multiple excluded directories, click **Add another Directory Exclusion**.

Note: Directory Exclusion Pattern only excludes sub-directories inside the scan targets. It cannot exclude archives or directories and contents inside an archive.

4.4 Dry Run Option

Located below the **Scan Memory Allocation** field is the Hub Jenkins **Dry Run** option. The **Dry Run** option allows you to perform scans without uploading the scan results to the server. It creates a JSON file of the scan results within the workspace, which is located in `HubScanLogs/{BUILD_NUMBER}/data/`. Note that project and version are still required.

Because **Dry Run** does not upload scan results to the server:

- No report is generated since the scan results are never sent to the Hub.
- Failure conditions are not run since the scan results are never sent to the Hub.

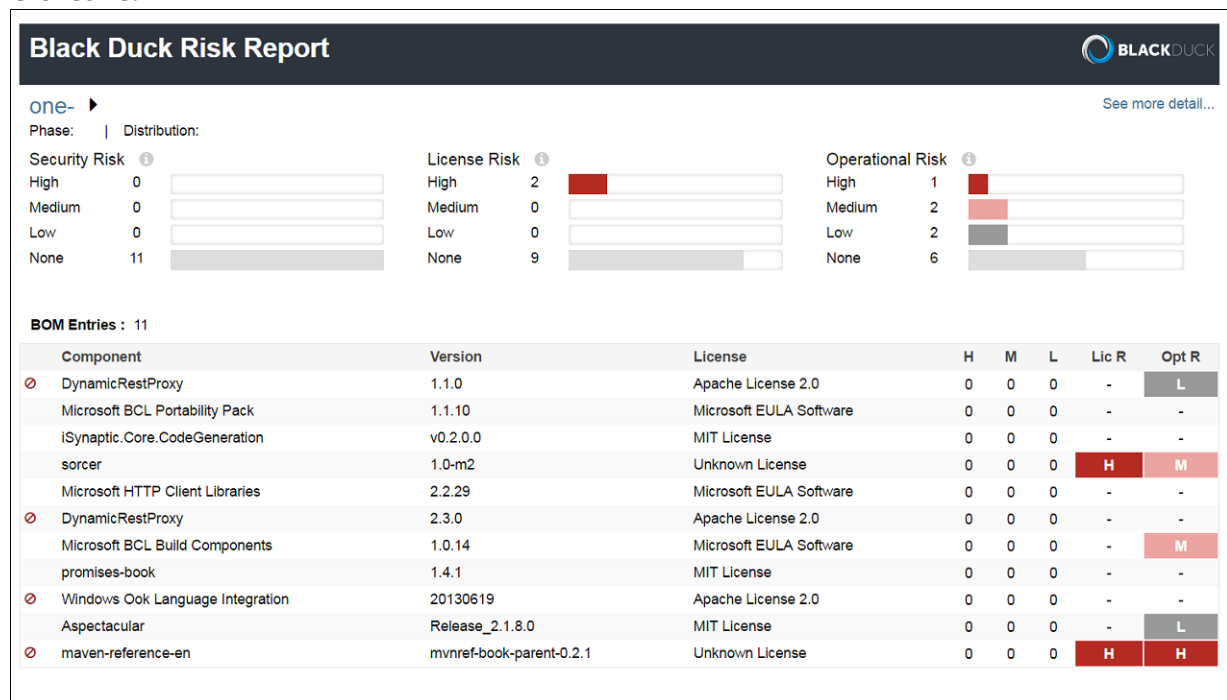
If you configure the scan to be a dry run, the following message appears in the logs if you have added the failure conditions: *Will not run the Failure conditions because this was a dry run scan.*

4.5 Configuring Hub Risk Reports

As of Hub Jenkins version 1.5.3 and higher, the risk report displays components that are in violation. To generate Hub risk reports within Jenkins, you must perform the following process.

✳ To show the Hub risk report in Jenkins:

1. Click **Advanced**.
2. Select **Generate Black Duck Risk Report**.
3. Specify the maximum amount of time to wait for the report. If the Hub Bill of Materials (BOM) for the specified version is not updated with the information from the scan within the specified time, a timeout is forced and the build is set to **Unstable** so as not to delay the build.
4. Enter the amount of memory to allocate to the Black Duck scan in megabytes. The default is 256MB.
5. Enter the target to scan.
6. To define more targets, click **Add another Scan target**. If you do not specify a target, the entire workspace is scanned.

7. Click **Save**.

4.6 Generating Reports

* To generate reports in Hub Jenkins:

1. Go to the configured job.
2. Click **Build Now**. The BlackDuck scans messages display in the console output of the build.
3. If you selected **Generate Black Duck Risk Report**:
 - a. Click the build; the **Black Duck Risk Report** displays in the sidebar.
 - b. Click **Black Duck Risk Report** to open a page with a summary of the Black Duck Risk Report for the specified version. The results display for the BlackDuck Scans in the Hub server.

Black Duck Risk Report

James Test > 2

Phase: In Planning | Distribution: External

Security Risk
 High 0
 Medium 0
 Low 0
 None 8
 Calculated risk based on known vulnerabilities.

License Risk
 High 1
 Medium 0
 Low 0
 None 7
 Calculated risk based on open source license use.

Operational Risk
 High 1
 Medium 0
 Low 3
 None 4
 Calculated risk based on tracking overall open source component activity.

Bom Entries : 8

Component	Version	License	H	M	L	Lic R	Opt R
Aspectacular	Release_2.1.8.0	MIT License	0	0	0	-	L
Debian Java packages Team	5.6.0	Apache License 2.0	0	0	0	-	-
DynamicRestProxy	1.1.0	Apache License 2.0	0	0	0	-	L
ISynaptic.Core.CodeGeneration	v0.2.0.0	MIT License	0	0	0	-	-
Kotlin	build-1.0.0	Apache License 2.0	0	0	0	-	L
maven-reference-en	mvnref-book-parent-0.5	Unknown License	0	0	0	H	H
Microsoft HTTP Client Libraries	2.2.29	Microsoft EULA Software	0	0	0	-	-
promises-book	1.4.1	MIT License	0	0	0	-	-

Page generated: Mar 10, 2016 12:20:47 PM Jenkins ver. 1.642.1

4.7 Hub Failure Conditions

Black Duck Hub Failure Conditions: This functionality is new for Hub version 3.0.0 and higher, and checks the BOM to verify if any components are in violation of a policy. If so, it fails the build. This function is not available if the server you defined does not support policies.

* To configure failure conditions:

1. Go to job configuration.
2. Add post-build action.
3. Select **Black Duck Hub Failure Conditions**.
4. Select **Fail the Build for Hub Policy violations**.

Add another Scan target

Black Duck Hub Failure Conditions (Hub 3.0+)

Requires the policy management module in the Hub.

☒ Fail the Build for Hub Policy violations

Add post-build action

Save

Apply

✳ To use failure condition functionality once configured:

1. Click **Build Now** on the job that is configured with failure conditions.
2. The build is set to *Failure* if the overall BOM status is *In Violation*, meaning there are components in violation of a defined policy.

Post-build Actions

Black Duck Hub Integration

Project Name

James Test

This Project exists on the Hub Server : http://integration-hub.blackducksoftware.com

Project Version

2

This Version exists in the Project : James Test

Phase

In Planning

Distribution

External

Create Project/Version

Advanced...

Scan Memory Allocation

4096

Scan Target

The scan target is empty, the entire workspace will be scanned

Delete Scan target

Add another Scan target

Black Duck Hub Failure Conditions

☒ Fail the Build for Hub Policy violations

The Hub server configured does not have support for Policies.

Delete

Add post-build action

Save

Apply

Things to keep in mind when using failure conditions:

- The Black Duck Hub failure conditions can only be configured if the Black Duck Hub Integration is also configured.
- The Black Duck Hub failure conditions must be configured after the Black Duck Hub Integration step in the job configuration. Post-build actions run in the order they are configured.
- If the Hub server configured in the global configuration is older than version 3.0.0, then the **Black Duck Hub Failure Conditions** does not display in the list of post-build actions, as this version does not support policies.
- If the Black Duck Hub failure conditions are configured, and then the global configuration changes to use a Hub server that does not support policies, an error displays. If a build is run, it is set to **Unstable**.
- If a job is configured with Black Duck Hub failure conditions but the check box is not selected, then the build is set to **Unstable** since the failure conditions are not configured to do anything.

4.8 Setting Configurable Build State

You can set a configurable build state for build failure conditions. This enables you to pass or fail the build, and log the results.

✳ To set a configurable build state:

1. Go to job configuration.
2. Add post-build action.
3. Select **Black Duck Hub Failure Conditions**.
4. Select **Fail the Build for Hub Policy violations**.
5. Using the Build State on Failure Condition selector, choose the failure condition. Options are:
 - a. Success = Does not fail the build.
 - b. Unstable = Sets the build result to unstable.
 - c. Failure = Sets the build result to failure.

☒ Fail the Build for Hub Policy violations

Build State on Failure Condition: Success

If the Failure Condition is met, then the Build will be set to this state on completion.

Success = Will not fail the Build

Unstable = Will set the Build Result to Unstable

Failure = Will set the Build Result to Failure

(from [Black Duck Hub Plugin for Jenkins](#))

Chapter 5: Hub Jenkins Best Practices

This section provides best practices information for using and configuring Jenkins, based on conducted research.

5.1 Permissions

The Hub *Code scanner* role must be assigned to a Hub user to use the Hub Jenkins plugin.

5.2 Post Build Action Blocking

Post-build actions use a `BuildStepMonitor` as returned by `getRequiredMonitorService` in the post-build action. This defines how jobs are queued, based on the post-build action.

If this method returns the value of `BuildStepMonitor.BUILD`, then another job that must execute the same post-build action is queued. This job waits for the previous job to complete its execution before it can execute (using the same post-build action). This provides a degree of safety if concurrent builds are executed; however, it blocks multiple jobs from running synchronously. This may not be desirable because the job is meant to build independent projects, and the scans should be done in parallel.

If this method returns the value of `BuildStepMonitor.NONE`, then any job executing the same post-build action runs immediately when possible. This is the current behavior of the Black Duck scan post-build action. There is no blocking provided; therefore, no waiting occurs. This procedure allows for concurrent execution.

Caution: This can be dangerous because the Jenkins workspace for the job could be overwritten by another execution of the same job.

5.3 Sharing Workspace with a Downstream Job

Note: Using a custom workspace can be dangerous. The project can perform a cleaning of the workspace, meaning it deletes the files in the workspace. Therefore, the workspace folder should not be a folder containing files required for the OS, or files to be saved.

If a job needs to share a workspace with a downstream project, then one way of achieving this is by configuring the downstream project using a custom workspace. The value in the directory text field for the custom workspace can be a path to a specified location. Or, for greater flexibility, the value of the directory text field can be an environment variable. Then an upstream project can define the variable with the value of the workspace used by the upstream project. Each job has a `$WORKSPACE` environment variable defined by Jenkins. The upstream project can set an environment variable to the value of its `$WORKSPACE` variable in order to share the workspace between the jobs. For example, `CUSTOM_WORKSPACE=$WORKSPACE`. This is one way of sharing the workspace between jobs.

The Jenkins plugins *Multijob* and the *Parameterized Trigger* provide additional functionality to make it easier to set the environment variables available to downstream projects. Note the *Multijob* Jenkins plugin has a dependency on the *Parameterized Trigger* plugin.

You can access the *Multijob* plugin at: <https://wiki.jenkins-ci.org/display/JENKINS/Multijob+Plugin>.

5.4 Parameterized Job Configuration

To invoke the Hub scan as a separate job, you must configure a parameterized job to invoke this job as a downstream job. The *Parameterized Trigger* plugin is required for triggering parameterized jobs.

At the top of a build configuration are check boxes for configuring the job. To create a parameterized job you must define parameters. To define parameters, select the **This build is parameterized** check box.

* To create a new freestyle project:

1. Select **This build is parameterized**, and define the following parameters:
 - a. HUB_APP_NAME Default value = <empty>
 - b. HUB_APP_VERSION Default value = <empty>
 - c. HUB_SCAN_TARGET_1
 - d. HUB_SCAN_TARGET_N

☒ This build is parameterized

String Parameter

Name

Default Value

Description

[Safe HTML] [Preview](#)

Delete

String Parameter

Name

Default Value

Description

[Safe HTML] [Preview](#)

Delete

String Parameter

Name

Default Value

Description

The parameter default values are defined as empty strings because the usage of `HUB_APP_NAME` and `HUB_APP_VERSION` cause the execution of the Hub scan post build action to fail. Using the `CUSTOM_WORKSPACE` variable assumes that the workspace between the upstream job and the parameterized job will be shared. Extra caution must be taken to assure the integrity of the workspace during the invocation of the parameterized job.

2. Configure the job to execute concurrent builds.
3. Select the **Execute concurrent builds if necessary** check box. This allows multiple upstream jobs to invoke this parameterized job so that each instance of the parameterized job can execute independently of the other invocations of the job.
4. Configure the workspace for the project in order to share the workspace with the upstream project.
 - a. Select the **Use custom workspace** check box.
 - b. In the **Directory** field, enter the variable `${CUSTOM_WORKSPACE}`.
 - c. In the **Display Name** field, type the project name.
5. Configure the Hub scan post build action:

- a. In the **Project Name** field, enter the previously-defined variable `${HUB_APP_NAME}`.
- b. For the **Project Version**, enter the previously-defined variable `${HUB_APP_VERSION}`.
- c. Configure the **Advanced** settings as appropriate.
- d. Add 1 to N scan targets.
- e. For each added scan target, set the value to a variable that is passed by the upstream project. For example, `${HUB_SCAN_TARGET_1}`, `${HUB_SCAN_TARGET_2}`, ..., `${HUB_SCAN_TARGET_N}`.

5.5 Downstream Job Configuration

Downstream job configuration is one of the ways you can configure a job to invoke a parameterized build. There are two methods:

- Configure a downstream job by using the trigger parameterized build post build action.
- Configure a Multijob.

The method you choose depends on your preferences and workflows.

In this scenario, you will configure a job to invoke a parameterized job in the post build action of the job, and you will create a job that builds the source code. Then a parameterized job is invoked in the post build action of the job to perform the scan.

Note: This scenario utilizes the sharing of a workspace between the jobs and a parameterized build. Therefore, extra care is needed when configuring this type of Job.

Configure a Parameterized Job for Hub Scanning

First, a parameterized job that can be invoked by another job must be defined. Refer to the *Parameterized Job Configuration* section for more information.

✳ To configure the build job:

1. Configure a Jenkins job to build a project.
2. Click **Add a post-build Action**.
3. Select **Trigger parameterized build on other projects**.
4. Select the previously-defined parameterized Hub build job.
5. Click **Add Parameters**.
6. Select **Build on Same Node**. Since the workspace is being shared by defining the `CUSTOM_WORKSPACE` variable, the build of the downstream project must occur on the same node as the current project.
7. Select **Predefined Parameters**.
8. Define `CUSTOM_WORKSPACE=${WORKSPACE}`. This is the parameter containing the path to the workspace in use. The `${WORKSPACE}` variable contains the current workspace of the job running;

therefore, we want to pass this path to the job downstream for it to use the same workspace as this job.

9. Define `HUB_APP_NAME` (the name of the Hub application).
10. Define `HUB_APP_VERSION` (the version of the application to scan).
11. Define `HUB_SCAN_TARGET_1` (the path of the first scan target for the Hub).
12. Define `HUB_SCAN_TARGET_N` (the path of the Nth scan target for the Hub).

Post-build Actions

Trigger parameterized build on other projects

Build Triggers

Projects to build:

Trigger when build is:

Trigger build without parameters: ☐

Build on the same node

Predefined parameters

Parameters

```
CUSTOM_WORKSPACE=${WORKSPACE}
HUB_APP_NAME=PSTestApp
HUB_APP_VERSION=0.1
HUB_SCAN_TARGET_1=/mvnbook-examples-1.0/ch-multi-spring/simple-parent/simple-command
```

Delete

5.6 Multijob Configuration

This section discusses how to configure the jobs to handle defining a single parameterized job to run a Hub scan. This requires sharing the workspace between jobs which can cause instability in your builds if a job modifies the files in a workspace required by a downstream job.

Note: This scenario utilizes the sharing of a workspace between the jobs and a parameterized build. Therefore, extra care is needed when configuring this type of job. The parameterized job must be run in a separate phase than the phase to build the source code. This is because jobs in the same phase can be executed concurrently. Adding a build step to define an additional phase in the Multijob ensures that the first phase (the build phase), completes before the phase invoking the parameterized job.

First, a parameterized job that can be invoked in one of the Multijob phases must be defined. Refer to the *Parameterized Job Configuration* section for more details.

* To configure a Multijob:

1. Create a new Multijob project.

Item name

☐ **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☒ **MultiJob Project**
MultiJob Project, suitable for running other jobs

☐ **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

☐ **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

☐ **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

☐ **Copy existing item**
Copy from

2. In the **Build** section, click **Add build step**.
3. Select **Multijob Phase**.
4. Type a name in the **Phase Name** field; for example, *Build Phase*.
5. Click **Add jobs...**, and select the job to build the source code.
6. Click **Add build step**, and select **Multijob Phase**.
7. Type a name in the **Phase Name** field; for example, *Scan Phase*.
8. Click **Add jobs...**, and select the name of the parameterized job defined in the **Parameterized Job Configuration** section.
9. Click **Advanced**, and click **Add Parameters**.
10. Select **Build on the same node**.
11. Select **Predefined parameters**.
12. Define `CUSTOM_WORKSPACE=${WORKSPACE}`. This is the parameter containing the path to the workspace in use. The `${WORKSPACE}` variable contains the current workspace of the running job; therefore, pass this path to the job downstream so it can use the same workspace as this job.
13. Define `HUB_APP_NAME` as the name of the Hub application.
14. Define `HUB_APP_VERSION` as the version of the application to scan.
15. Define `HUB_SCAN_TARGET_1` as the path of the first scan target for the Hub.
16. Define `HUB_SCAN_TARGET_N` as the path of the Nth scan target for the Hub.

MultiJob Phase

Phase name:

Phase jobs:

Job

Job name:

Kill the phase on:

Build only if SCM changes: ☐

Disable: ☐

Abort all other job: ☒

Current job parameters: ☒

☐ Enable retry

☐ Enable condition

Build on the same node

Predefined parameters

Parameters:

```
CUSTOM_WORKSPACE=${WORKSPACE}
HUB_APP_NAME=PSTestApp
HUB_APP_VERSION=0.1
HUB_SCAN_TARGET_1=mvnexusbook-examples-1.0/ch-multi-spring
```

Add Parameters

Note: You can either define the build steps in the Multijob to invoke and build the source code, or you can define separate parameterized jobs to build the source code. If you specify separate parameterized jobs to build the source code, they must share the same workspace as the scan job; therefore, they must share the same workspace and build on the same node as the scan.

5.7 Best Practices: Relative Paths

For Hub Jenkins versions 2.0.0 and earlier, all targets were relative to the workspace. As of version 2.1.0 and higher: if you provide an absolute path, Hub Jenkins uses the absolute path rather than making it relative to the workspace.

Examples

Versions 2.0.0 and earlier:

```
Target Resolves to
/test/ ${WORKSPACE}/test
subDir/ ${WORKSPACE}/subDir
```

Versions 2.1.0 and higher:

```
Target Resolves to  
/test/ /test  
subDir ${WORKSPACE}/subDir
```

Chapter 6: Troubleshooting the Hub Jenkins Plugin

If an error message is generated that states *During development and testing the following errors were encountered*, use the following solutions:

- If you try to use Java 6 instead of Java 7, instead of getting an *Unsupported major:minor version error* message, the plugin sometimes throws a false *java.lang.OutOfMemoryError: Java heap space* message instead.
- If you get a message that reads *Service Unavailable*, either the Hub server can't be reached, or the request to the server is invalid. Contact your Hub server administrator.
- If you get a *Precondition failed* error message, then the request to the server is invalid. Verify that your global configuration is correct, and verify that the job configuration is correct. If you are still getting this message after you have checked your configuration, contact your Black Duck technical account manager.
- If you get a *Not Found (404) - Not Found* error message, then the request to the server is invalid. Contact your Black Duck technical account manager.

Tip: After major releases of Hub, check for updated versions of your Black Duck plugins. Changes to the APIs, schema, and SDK versions may require updated versions of the integration plugins.

- If you try to use the Hub Jenkins integration, and you configure a job with a project and version, and that project already exists but the current Hub user is not assigned to it, then the following errors display:
 - In the job configuration **Project Name** field, a notification displays *This project does not exist on the Hub Server*. Clicking **Create project/version** displays a message reading *This version may already exist. com.blackducksoftware.integration.hub.exception.BDRestException: There was a problem creating this Hub project. Error Code: 412.*
 - If you run the build, the following displays:

Status : 412

```
Response : {"errorMessage":"project name already exists","arguments":
{"fieldName":"name"},"errors":[{"errorMessage":"project name already
exists","arguments":{"fieldName":"name"},"errorCode":"{central.constraint_
violation.project_name_duplicate_not_allowed}"},"errorCode":"
{central.constraint_violation.project_name_duplicate_not_allowed}"]}
```

Problem creating the project.

Assigning the current user to the existing project with this name resolves the issue.

Chapter 7: Hub Jenkins Plugin Release Notes

Changes in Release 2.1.0

New Features

- To log a ticket for the Black Duck Hub Jenkins plugin, you must now log a ticket through the public Jenkins JIRA, and log a ticket against the Black Duck Hub Jenkins plugin.
- Hub Jenkins now features the ability to set a configurable build state.
- Added functionality for Code Locations.
- Added functionality for cleaning up logs for successful scans.
- Added functionality for directory exclusion patterns.

Changes in Release 2.0.0

Issues Resolved

- Addressed an issue wherein scanning the Hub user interface using the Jenkins scanner may result in an illegal state exception.
- Addressed an issue wherein testing a Hub connection may result in a exception error.

Changes in Release 1.5.4

- Addressed an issue wherein Hub scans may fail and display an *Unstable* error message.
- Improved performance with new functionality which enables you to select only your desired report sections prior to running reports.
- Step API dependency is now for Step API versions 2.3 and higher.
- Addressed an issue wherein the **Test Connection** procedure generated an incorrect error message.

Changes in Release 1.5.3

- Addressed an issue wherein the risk report is updated to show which components violate a policy.
- Hub server responses are now shown at the Global Configuration level.
- Addressed an issue wherein an error for **Scan Memory Allocation** when under 256MB displayed an incorrect error message.
- Pipeline plugin functionality is now supported in Jenkins versions 1.580.3 and higher.

Changes in Release 1.5.2

- Addressed an issue wherein integrations using the Black Duck Hub plugin were unable to establish a connection to internal Black Duck Hub installations during a scan.
- Addressed an issue wherein Security Risk counts with more than four digits were not displaying correctly.

Changes in Release 1.5.1

- A scan now creates a new version, even if that version already exists.
- Migrating from a version 1.4.0 job configuration to a version 1.5.1 job configuration is now successful.
- The level of logging is now configurable.

Changes in Release 1.5.0

- Initial Open Source release.
- Updating to use the Hub-common changes for public APIs.

Changes in Release 1.4.1

- Black Duck Failure Conditions are fixed; they are now added after the Black Duck Hub integration.
- Failure Conditions now work even if the project name and version contain variables.
- Failure Conditions now wait until the Bill of Materials is updated before checking the policy status.
- The field **Maximum time to wait for report (in minutes)** is changed to **Maximum time to wait for BOM update (in minutes)**.

Changes in Release 1.4.0

- Auto install of the Black Duck Hub Scanner (CLI).
- Ability to fetch a BOM report from Hub into Jenkins for display and archiving.
- Ability to fail the Jenkins build if a component fails Hub policy.
- Global network timeout for the Hub connection is now configurable.
- Update of the plugin through the Jenkins update site:
 - Independent of Hub releases.
 - Signaled in the Jenkins plugin management.

Changes in Release 1.3.7

Release 1.3.7 was a maintenance release.

- Fixed an issue where builds running on slaves log fewer messages to the console log than when the builds are running on master.

Changes in Release 1.3.6

Release 1.3.6 was a maintenance release.

- Improved compatibility with non-Oracle Java Runtime Environments (JREs).
- Improved determination of the local host name.

Changes in Release 1.3.5

Release 1.3.5 was a maintenance release.

- Fixed an issue with code locations being mapped to multiple projects.
- Fixed an issue with scanning more than 10 code locations in a single job.
- Fixed an issue wherein the command line interface fails if the Java Virtual Machine is a version lower than 1.7.

Changes in Release 1.3.3/1.3.4

Release 1.3.3/1.3.4 was a maintenance release.

- Fixed an issue where an invalid URL succeeds in test connection, but fails in the job run.
- Fixed an issue so that the log directory option works with Hub 2.1.5, 2.2, and later versions.

Changes in Release 1.3.2

Release 1.3.2 was a maintenance release.

- Fixed an issue where the command line interface login failed on Windows operating systems.

Changes in Release 1.3.1

Release 1.3.1 was a maintenance release.

- Fixed an issue where if, on a rescan, the generation of the log file fails, the plugin displays the previously generated log file.
- Fixed an issue where the incorrect port number was being passed to the Hub command line interface (CLI) when the Jenkins port value was null (no port number entered).
- Fixed an issue where the project name suggestions were listing KnowledgeBase projects. The project name suggestions now list only internal projects.

7.1 Hub Jenkins Known Issues

The following are known issues for the Hub Jenkins plugin.

- If more than one process tries to perform the CLI install at the same time to the same directory (on the same machine), then the processes collide with each other and start deleting files while another creates them.
 - Workaround: When a process finds that the currently installed CLI must be updated (downloaded for the first time or to change the current files), then it should create a `.locked` file (or a similar and appropriate file name), then perform the update, and remove the `.locked` file when the update is complete. If another process starts to perform the install, it should first check for the `.locked` file, if it exists, then it should wait until the `.locked` file is deleted before performing its check.

Chapter 8: Black Duck Support

If you have questions or find issues, contact Black Duck Software.

For the latest in web-based support, access the Black Duck Software Customer Support Web Site:
<https://www.blackducksoftware.com/support/contact-support>

To access a range of informational resources, services and support, as well as access to Black Duck experts, visit the Black Duck Customer Success portal at:
<https://www2.blackducksoftware.com/support/customer-success>

You can also contact Black Duck Support in the following ways:

- **Email:** support@blackducksoftware.com
- **Phone:** +1 781.891.5100, ext. 5
- **Fax:** +1 781.891.5145
- **Standard working hours:** Monday through Friday 8:00 AM to 8:00 PM EST

Note: Customers on the **Enhanced Customer Support Plan** are able to contact customer support 24 hours a day, 7 days a week to obtain Tier 1 support.

If you are reporting an issue, please include the following information to help us investigate your issue:

- Name and version of the plugin.
- Black Duck product name and version number.
- Third-party integrated product and version; for example, Artifactory, Eclipse, Jenkins, Maven, and others. For Black Duck Hub, only Jenkins, TeamCity, and Bamboo is supported.
- Java version.
- Black Duck KnowledgeBase version, where applicable.
- Operating system and version.
- Source control management system and version.
- If possible, the log files, configuration files, and Project Object Model (POM) XML files.

8.1 Training

Black Duck training courses are available for purchase. Learn more at
<https://www.blackducksoftware.com/services/training>.

View the full catalog of our online offerings: <https://www.blackducksoftware.com/academy-catalog>.

When you are ready to learn, you can log in or sign up for an account:
<https://www.blackducksoftware.com/academy>.

8.2 Services

If you would like someone to perform Black Duck Software tasks for you, please contact the Black Duck Services group. They offer a full range of services, from planning, to implementation, to analysis. They also offer a variety of training options on all Black Duck products. Refer to <https://www.blackducksoftware.com/services/> for more information.