BLACKDUCK

Installing, Configuring, and Using the Hub Plugin for Jenkins

Version 1.5.0

This edition of the *Installing, Configuring, and Using the Hub Plugin for Jenkins* refers to version 1.5.0 of the Hub Plugin.

This document created or updated on Monday, April 18, 2016.

Please send your comments and suggestions to:

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

Contents

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 2.4 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

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 Installation Overview

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

1.3.1 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.

1.3.2 Downloading and Installing the Hub Jenkins Plugin

Download the Jenkins plugin from the Hub online Help, then install the plugin using Jenkins.

✱ 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.

1.4 Configuring the Hub Jenkins Plugin

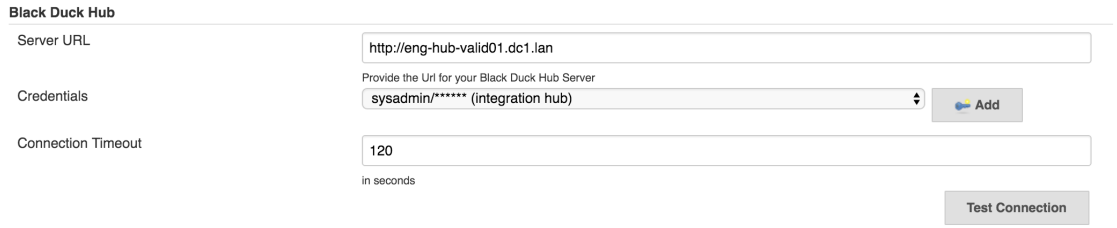
✱ To configure the Hub Jenkins plugin:

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:

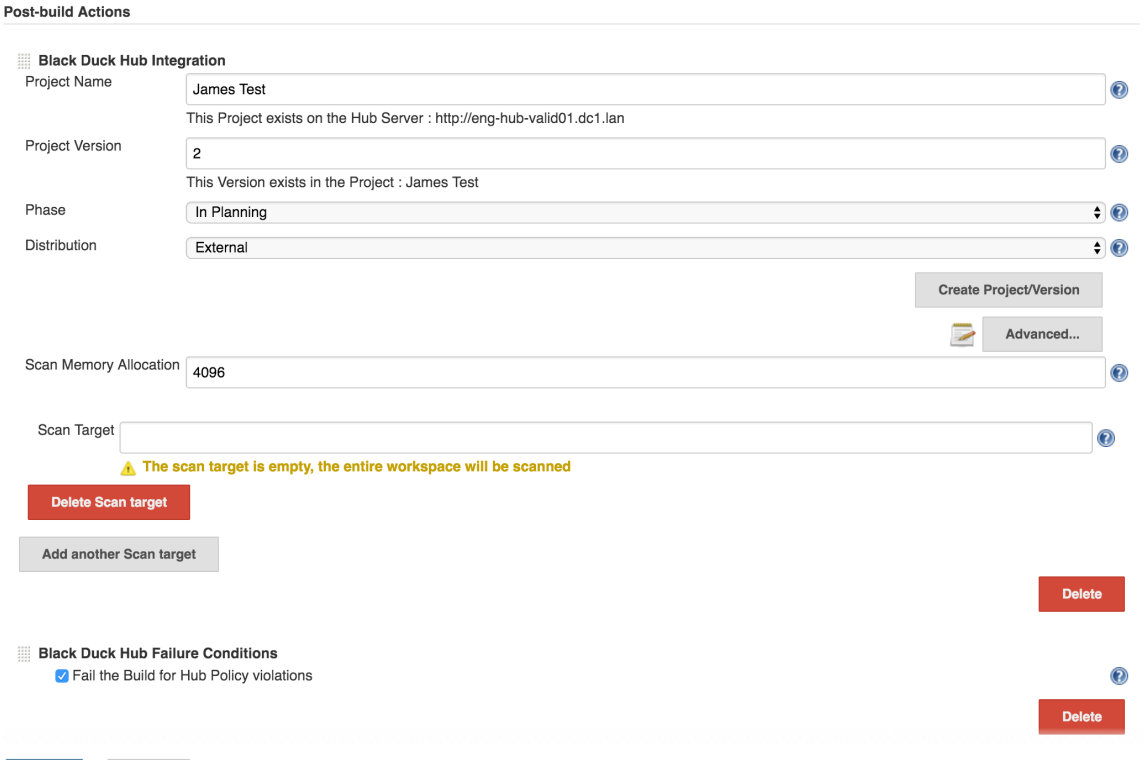
    ```
    http://hub.blackducksoftware.com:8080
    ```

    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.



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.



   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**, **Project Version**, and **Scan Target** fields all 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. If the project or release does not exist, you can create the project or release by clicking **Create Project/Version**.

Note: Leave the **Project Name** and **Project Version** field empty if you do not want the scans to be mapped to a particular release. The resulting scans for the specified scan targets are automatically mapped to the specified project release.

g. From the menu in the **Phase** field, select the current development phase of this project:

- Planning
- Development
- Released
- Deprecated
- Archived

h. From the menu in the **Distribution** field, select the method by which this version of the project is being released:

- External
- SaaS
- Internal

Note: The **Phase** and **Distribution** fields are only available when you are creating a new version. They cannot be edited for existing versions.

i. You can display the Black Duck Hub risk report within Jenkins.

j. 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.

k. In the **Scan Target** field, enter the path to the file you want to scan.

l. 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.

1.5 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.

1.6 Using the Hub Jenkins Plugin

✱ **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.

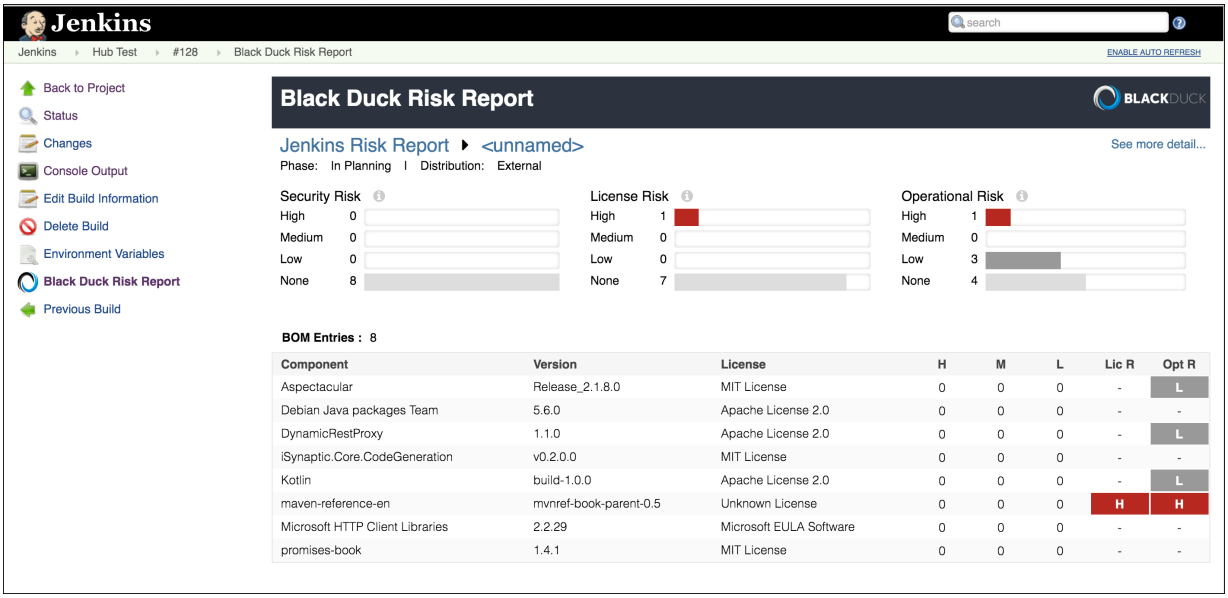
Alternatively, you can also view the logs from the BlackDuck scans in the directory `{SCAN_HOME}/lib/log`.

1.6.1 Configuring Hub Risk Reports

To generate Hub risk reports within Jenkins, you must perform the following configuration.

✱ 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 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**.



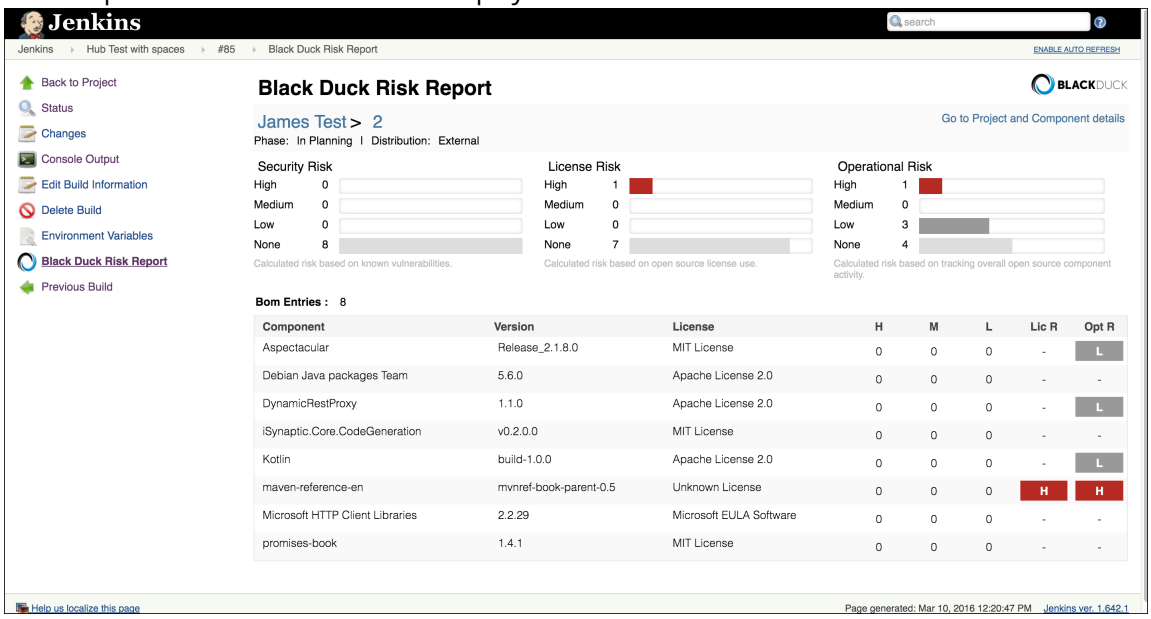
1.6.2 Generating Reports

✱ 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.

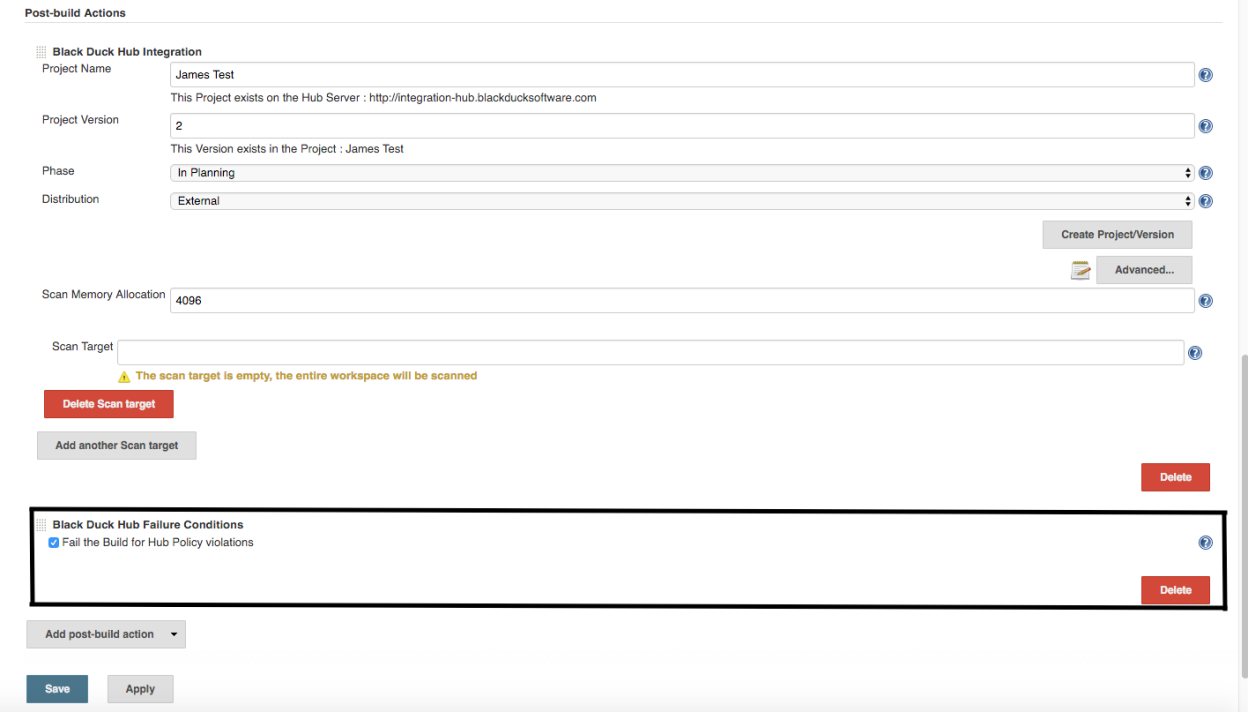


1.6.3 Hub Failure Conditions

Black Duck Hub Failure Conditions: This functionality is new for Hub version 3.0.0, 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.

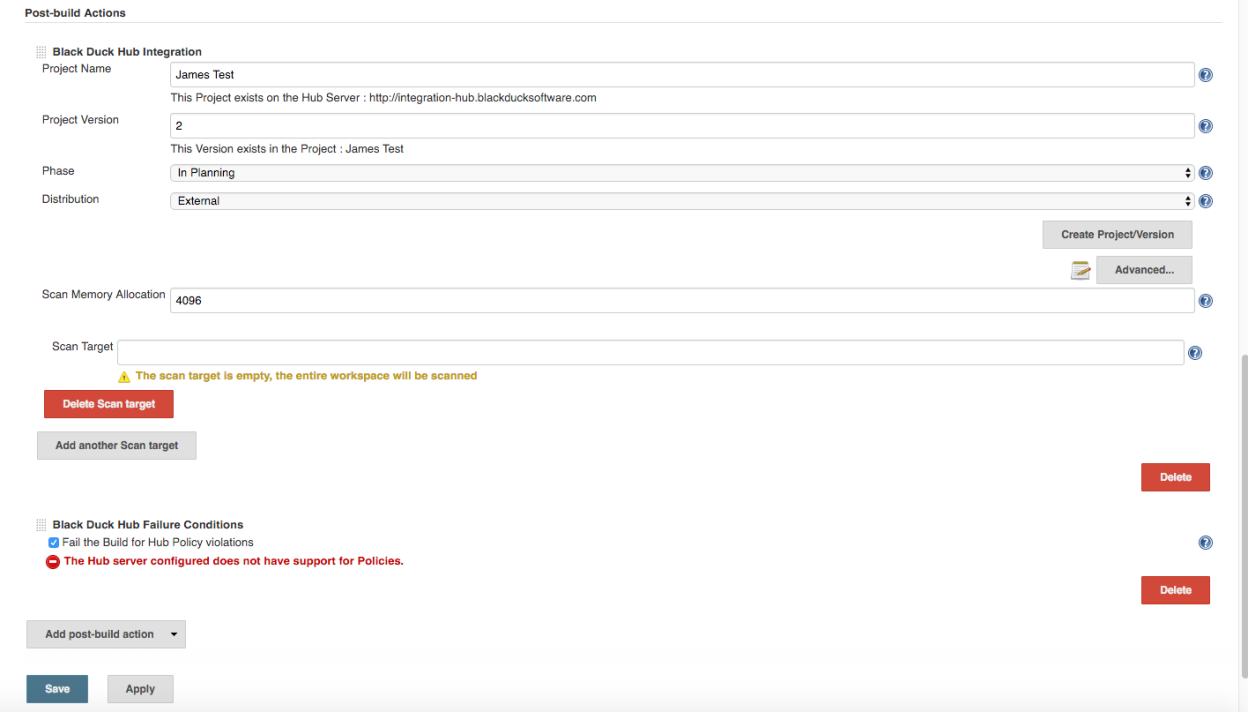
✱ To configure failure conditions:

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**.



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.



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.

1.7 Hub Jenkins Best Practices

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

1.7.1 Configuring Projects

This section describes information regarding configuring jobs in Jenkins.

1.7.2 Hub Integration

This section describes some of the configuration items to be set when configuring the build environment

for Hub integration.

1.7.3 Permissions

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

1.7.4 Concurrency

This section describes concurrency and how it relates to the execution of jobs.

1.7.5 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.

1.7.6 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.

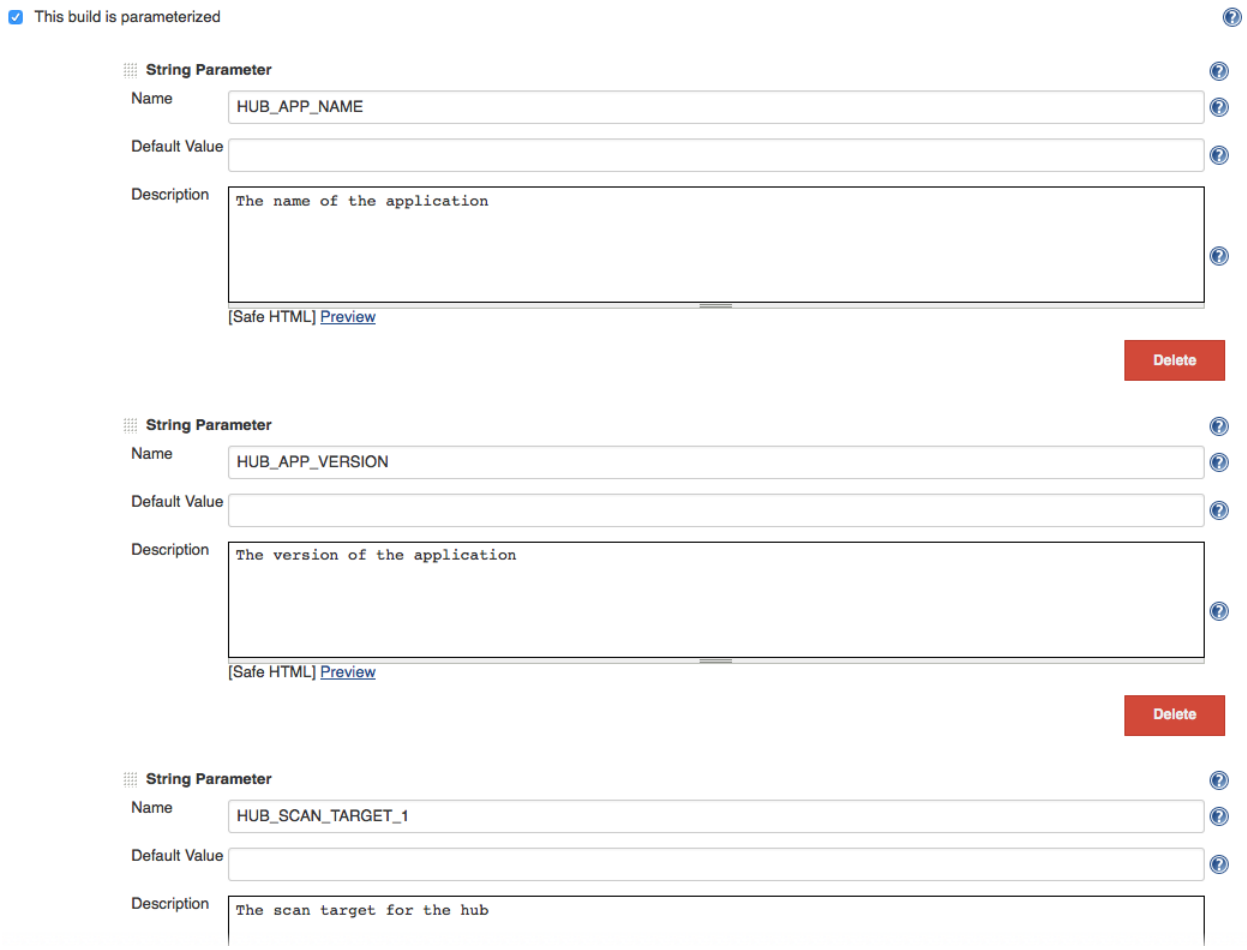
1.7.7 Parameterized Job Configuration

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`



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}`.

1.7.8 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.

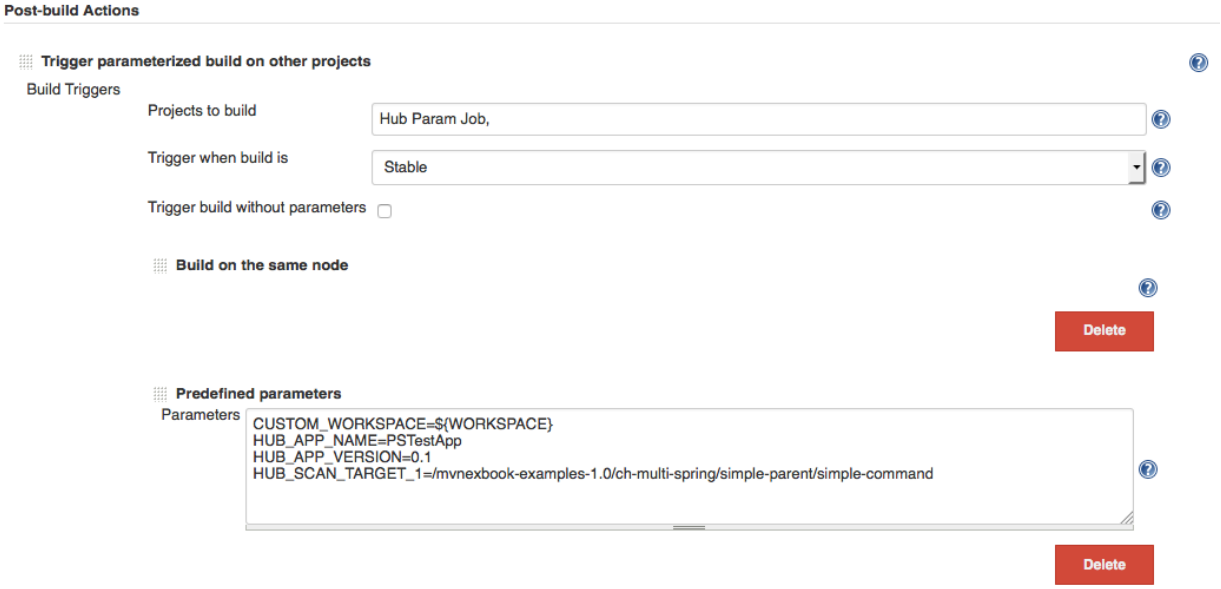
Note: This scenario utilizes the sharing of a workspace between the jobs and a parameterized build. Therefore, extra care is needed when configuring this type of Job.

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).



1.7.9 MultiJob Configuration

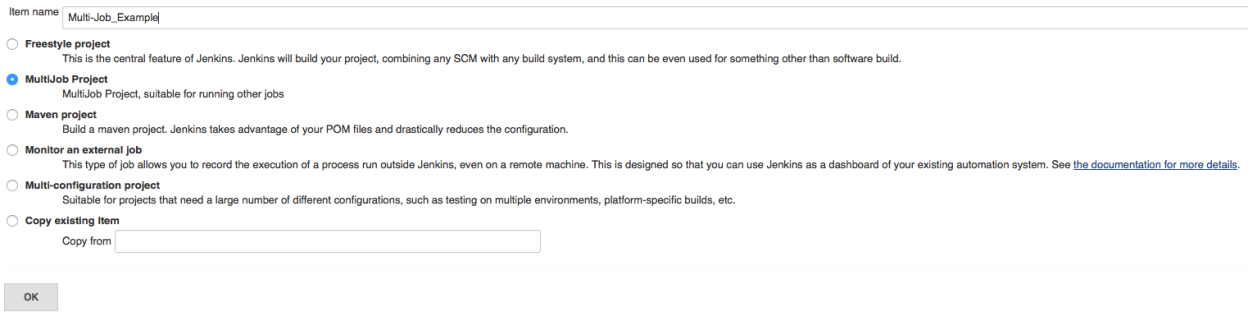
This section discusses how to configure the jobs to handle defining a single parameterized job to run a Hub scan. This requires sharing the workspace between jobs which can cause instability in your builds if a job modifies the files in a workspace required by a downstream job.

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.

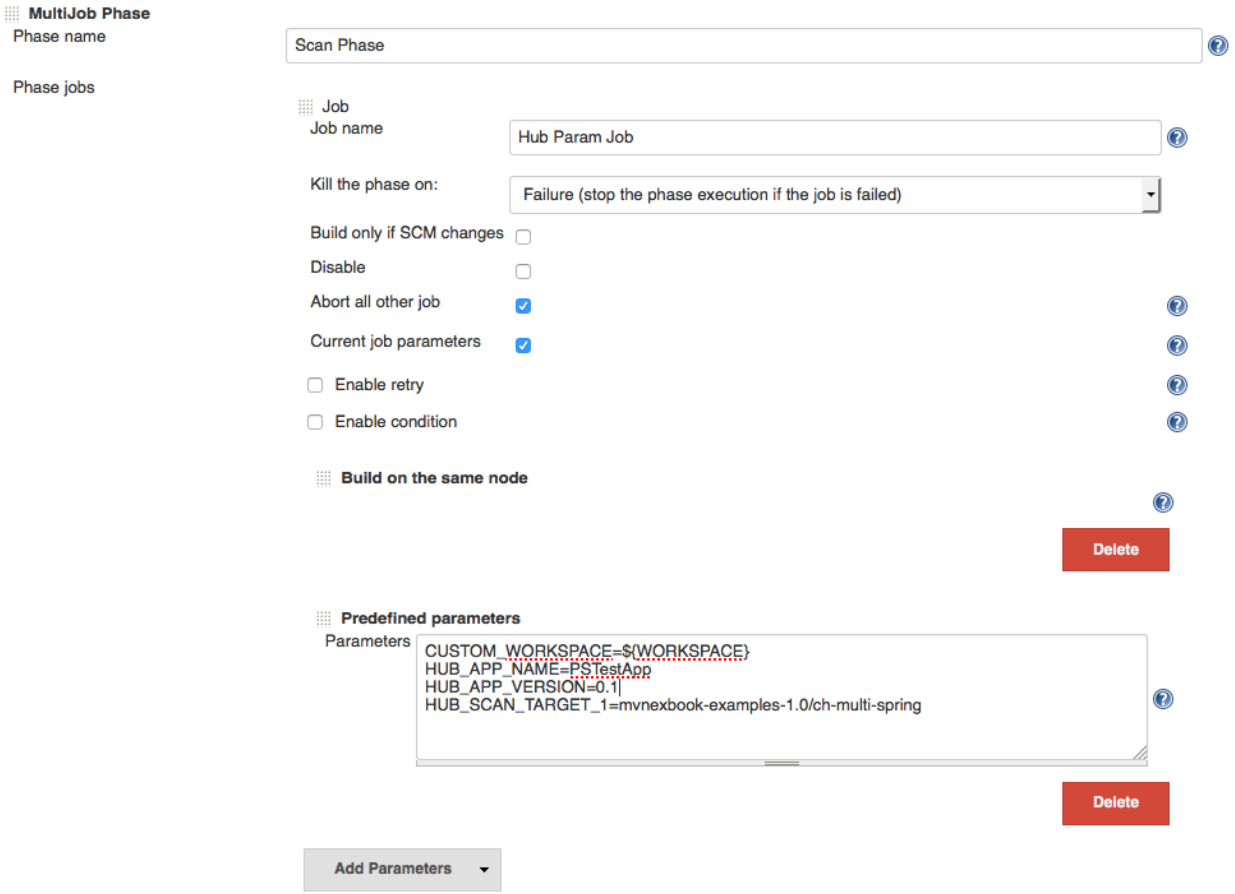
First, a parameterized job that can be invoked in one of the MultiJob phases must be defined. Refer to the *Parameterized Job Configuration* section for more details.

To configure a MultiJob:

1. Create a new MultiJob project.



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 N^{th} scan target for the Hub.



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.

1.8 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.

1.9 Hub Jenkins Plugin Release Notes

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.

1.10 Black Duck Support

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