

# Scanning Best Practices

Black Duck SCA 2026.7.2

Copyright ©2026 by Black Duck.

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 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.

03-11-2026

# Contents

**Preface....4**
Black Duck documentation....4
Customer support....4
Black Duck Community....5
Training....5
Black Duck Statement on Inclusivity and Diversity....5
Black Duck Security Commitments....6

**1. Scanning best practices....7**
About scanning tools, scans, and project versions....7
Black Duck Detect and the underlying tools used by it....7
The differences between full and rapid scanning....8
The relationship between scans and project versions....8
Scans and scan names....10
Scans and project versions for full scanning....10
Individual file matching for full scanning....10
When to use Black Duck Binary Analysis versus Black Duck....11
Configuring automated scans....11
Where/when in the build process to invoke the scan....12
Asynchronous versus synchronous full scans....13
Scan names, project versions, and versioning for full scanning....13
Projects using multiple branches....13
Scanning docker images....16
Signature scanning very large projects....16
Handling reusable modules (libraries)....16
Scanning when network connectivity is an issue....19
Poor scanning techniques....20
Including the commit ID or build ID in the Black Duck version or scan names....20
Keeping a history of versions on a development branch when full scanning....20
Avoid Parallel Scanning of the Same Code Location....20

**2. Rapid Scan Overview....22**

**3. Troubleshooting scanning issues....25**
Accidental full scan proliferation by folder paths which include build or commit ID....25
Solution....25
Accidental full scan proliferation by a build server farm....25
Solution....25

**4. Frequently Recommended Black Duck Detect Properties....26**

# Preface

## Black Duck documentation

The documentation for Black Duck consists of online help and these documents:

| Title | File | Description |
|---|---|---|
| Release Notes | release_notes.pdf | Contains information about the new and improved features, resolved issues, and known issues in the current and previous releases. |
| Installing Black Duck using Docker Swarm | install_swarm.pdf | Contains information about installing and upgrading Black Duck using Docker Swarm. |
| Installing Black Duck using Kubernetes | install_kubernetes.pdf | Contains information about installing and upgrading Black Duck using Kubernetes. |
| Installing Black Duck using OpenShift | install_openshift.pdf | Contains information about installing and upgrading Black Duck using OpenShift. |
| Getting Started | getting_started.pdf | Provides first-time users with information on using Black Duck. |
| Scanning Best Practices | scanning_best_practices.pdf | Provides best practices for scanning. |
| Getting Started with the SDK | getting_started_sdk.pdf | Contains overview information and a sample use case. |
| Report Database | report_db.pdf | Contains information on using the report database. |
| User Guide | user_guide.pdf | Contains information on using Black Duck's UI. |

The installation methods for installing Black Duck software in a Kubernetes or OpenShift environment are Helm. Click the following links to view the documentation.

- Helm is a package manager for Kubernetes that you can use to install Black Duck. Black Duck supports Helm3 and the minimum version of Kubernetes is 1.13.

Black Duck integration documentation is available on:

- https://sig-product-docs.blackduck.com/bundle/detect/page/integrations/integrations.html
- https://docs.blackduck.com/p/bridgeintegrations

## Customer support

If you have any problems with the software or the documentation, please contact Black Duck Customer Support:

- Online: https://community.blackduck.com/s/contactsupport
- To open a support case, please log in to the Black Duck Community site at https://community.blackduck.com/s/contactsupport.
- Another convenient resource available at all times is the online Community portal.

# Black Duck Community

The Black Duck Community is our primary online resource for customer support, solutions, and information. The Community allows users to quickly and easily open support cases and monitor progress, learn important product information, search a knowledgebase, and gain insights from other Black Duck customers. The many features included in the Community center around the following collaborative actions:

- Connect – Open support cases and monitor their progress, as well as, monitor issues that require Engineering or Product Management assistance
- Learn – Insights and best practices from other Black Duck product users to allow you to learn valuable lessons from a diverse group of industry leading companies. In addition, the Customer Hub puts all the latest product news and updates from Black Duck at your fingertips, helping you to better utilize our products and services to maximize the value of open source within your organization.
- Solve – Quickly and easily get the answers you're seeking with the access to rich content and product knowledge from Black Duck experts and our Knowledgebase.
- Share – Collaborate and connect with Black Duck staff and other customers to crowdsource solutions and share your thoughts on product direction.

Access the Customer Success Community. If you do not have an account or have trouble accessing the system, click here to get started, or send an email to community.manager@blackduck.com.

# Training

Black Duck Customer Education is a one-stop resource for all your Black Duck education needs. It provides you with 24x7 access to online training courses and how-to videos.

New videos and courses are added monthly.

In Black Duck Education, you can:

- Learn at your own pace.
- Review courses as often as you wish.
- Take assessments to test your skills.
- Print certificates of completion to showcase your accomplishments.

Learn more at https://blackduck.skilljar.com/page/black-duck or for help with Black Duck, select **Black Duck Tutorials** from the Help menu () in the Black Duck UI.

# Black Duck Statement on Inclusivity and Diversity

Black Duck is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our

engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

# Black Duck Security Commitments

As an organization dedicated to protecting and securing our customers' applications, Black Duck is equally committed to our customers' data security and privacy. This statement is meant to provide Black Duck customers and prospects with the latest information about our systems, compliance certifications, processes, and other security-related activities.

This statement is available at: Security Commitments | Black Duck

# 1. Scanning best practices

Black Duck provides a wide array of scanning technology for performing software composition analysis, identifying open source software (OSS) and other third-party components present in software, so that the risks associated with using these components can be managed more effectively. This guide provides best practices for scanning. It assumes that you have already been introduced to scanning with Black Duck and know the basics of how to perform scans.

This document includes information on:

- What scanning client tool to use when configuring scans
- What mode of scanning—full or rapid—to use when scanning
- What is asynchronous versus synchronous full scanning and when to use them
- How to configure automated scans (for example, in a build server or in a CI/CD context)
- How to optimize scanning for performance
- How to avoid common pitfalls

Readers should also check the Black Duck Documentation Portal for additional content pertaining to scanning best practices.

## About scanning tools, scans, and project versions

To make the most of the best practices it is important to understand some basic behaviors of scanning and how scan results are made available to users through project-versions.

### Black Duck Detect and the underlying tools used by it

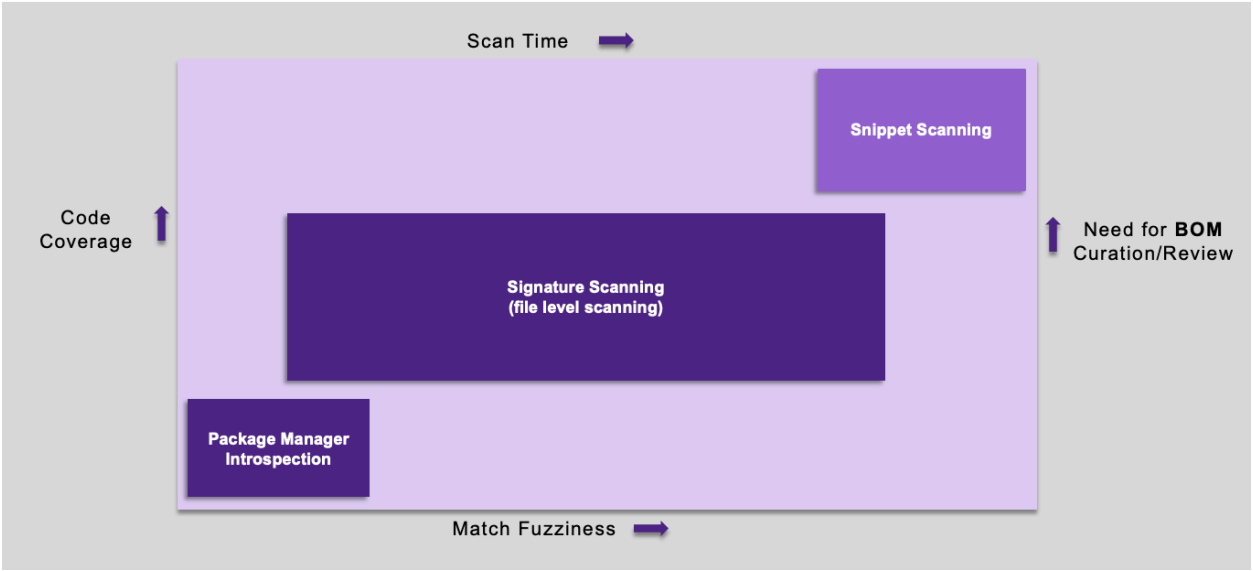
Black Duck Detect is the recommended client tool for scanning using Black Duck. It is packaged into a command line interface (CLI) and desktop GUI. Black Duck Detect makes it easier to set up and scan code bases using a variety of languages/package managers.

It uses several underlying tools to perform scanning including the Detector (for inspecting package manager dependencies), Signature Scanner (for inspecting the file system), and other tools. Options can be used to explicitly enable or disable these underlying tools, but by default Black Duck Detect will run both the Detector and Signature Scanner to provide good coverage on the code being analyzed.

The various scanning tools can increase code coverage, or fidelity, but at the expense of scan time and match fuzziness; see the diagram below. For instance, using package manager inspection, plus Signature Scanner, and snippet scanning provides the highest level of coverage but takes longer and will result in higher False Positive (FP) rates. Customers who want to optimize for speed may find using the package manager inspection provides sufficient coverage with very low/zero FP rates in a fraction of the time it takes to perform signature scanning.

Signature scanning is most often needed when there is no package manager inspection possible or when you need to be able to find use of components that are not declared by the package manager(s).

Snippet scanning is for the purpose of identifying code that was copied from some open source project and pasted into proprietary source files.




**Note:** Long-time Black Duck users may have grown accustomed to invoking the Signature Scanner directly. This approach is no longer recommended since it loses the power of employing the detector to find and leverage package manager files with declared dependencies.

## The differences between full and rapid scanning

Black Duck Detect offers two modes of scanning: full and rapid. Full scanning is used for source control branches (see the Gitflow discussion below) which are long-lasting or otherwise significant, such as release or integration branches. Rapid scanning is for transient or other less significant branches, such as a developer's working branch, where a persistent record of the results is not needed, and the primary goal is to provide the developer with feedback.

Use full scanning when the results should be persisted in the Black Duck server to track over time. Persisted results will be updated whenever new security violations or changes in license usage are recorded in the Black Duck KnowledgeBase (KB). Full scanning can also employ more detailed scanning techniques, including signature, snippet, and string search scanning.

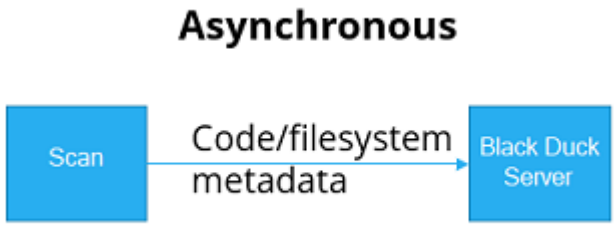
Use rapid scanning when persisting the data in Black Duck is not necessary. This is often the case when a developer wants quick results to determine if the versions of open source components included in a project violate corporate policies surrounding the use of open source. Rapid scans quickly return results, as they only employ package manager scanning and do not touch the Black Duck server database. Large customer deployments will often generate 10s of thousands of rapid scans a day.

## The relationship between scans and project versions

A scan occurs when a scan client (for example Black Duck Detect) is pointed at a folder (or a single `.tar` file).

There are two major sub-modes of full scanning: asynchronous and synchronous. Rapid scanning is always synchronous.

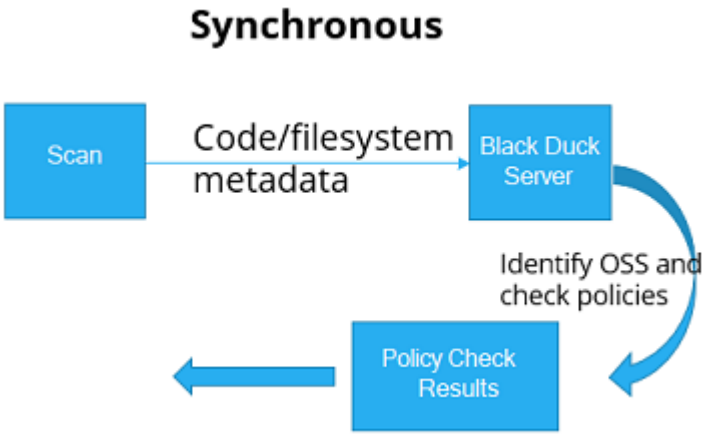
By default, full scans are asynchronous. This mode provides the best performance since the scan client does not wait for the results of the scan. An asynchronous scan works as described below




1. Scans generate metadata. Metadata includes:
   - Information from package managers regarding what open source components are used by the project
   - File hashes
   - Directory (folder) and file information
2. Metadata is uploaded to the Black Duck server by the scan client and, once the upload is complete, the scan client is finished.
3. Return/done (check for results later).

Therefore, an asynchronous scan is appropriate when performance is more important than obtaining the results immediately. Users can check for the results later using either the Black Duck UI or REST API. Black Duck Alert can also be used to push notifications of scan results to users.

In some cases, the results are needed immediately, for example, to “fail the build” for policy violations. In these cases, use a synchronous scan, as shown:




When using Black Duck Detect with either the **--detect.policy.check.fail.on.severities** or the rapid scanning properties set, a synchronous scan is performed which does the following:

1. The scan uploads metadata to the server.
2. The server processes the scan and assesses the results, including checking if any policies are violated.
3. For rapid scans, the list of policy violations is output to the command line. For full scanning, those results are available in the Black Duck UI or via the REST API.
4. The scan client (Black Duck Detect) waits for the results and exits appropriately, for example, the exit status is set to a non-zero value if any policies are violated.

So, an asynchronous full scan simply uploads the scan data and returns which is the best mode when optimizing performance for persisted scan results. A synchronous scan uploads the scan and waits for the results which is the right choice if the results are needed immediately.

## Scans and scan names

A scan gets a name which, by default, the Black Duck scan client generates. The derived name is formed by combining the hostname of the host running the scan with the full path of the folder being analyzed.

**Important:** One of the full scanning pitfalls is when the path being scanned includes a commit or build ID. In that case, every scan creates a completely new set of scan results in the database when the intent is usually to overwrite the previous scan. This can lead to a proliferation of saved scan results that can degrade system performance and make results in the mapped project version inaccurate. In most cases, it is not a good idea to include the build or commit ID in either the scan name or project version name. This problem only relates to full scanning. For rapid scanning, no results are saved in the Black Duck database. A notable exception is when developing on a single branch, such as often happens when developing web- or micro-services. In that case using commit or build ID in the scan and project version names may be necessary, but automated pruning of old versions/scans should be implemented also to keep the system in balance and uncluttered.

You can override the default and supply your own scan name by using the **--detect.code.location.name** property in Black Duck Detect (the recommended method) or the **--name** parameter if you are using the Signature Scanner CLI. Project and version names can be controlled using **--detect.project.name** and **--detect.project.version.name**, respectively.

For rapid scanning, use of the **--detect.code.location.name** property is recommended, as this allows scanning usage statistics to be grouped across non-notable source control branches for the same projects.

If you full scan again using the same name, the results of the previous scan are overwritten.

## Scans and project versions for full scanning

A scan is mapped to one, and only one, project version. A project version can have more than one scan mapped to it in the Black Duck database.

This allows mapping multiple, separate folders and the results of their scans into one aggregated project version. For example, you might have a library in one folder and the application that uses the library in another folder. You can scan the library's folder and the application folder separately and map their full scans to the same project version in Black Duck to see the aggregate of both the library and application.

Note that if you delete a project version, it does *not* delete the scans that were mapped to it; it simply unmaps the scans. If your intent is to clean the system of both the versions and the scans, you need to delete the project version and the scan. (Note that for rapid scans, no persistent BOM for such scans is created, so this problem does not occur.)

## Individual file matching for full scanning

Individual file matching is the identification of a component based purely upon the checksum information of a single file. Prior to Black Duck 2020.2.0, for a small set of file extensions (`.js`, `.apklib`, `.bin`, `.dll`, `.exe`, `.o`, and `.so`), regular signature scanning matched files to components based upon a checksum match to the one file. Unfortunately, this matching was not always accurate and produced a fair number of false positives. These false positives required you to spend additional effort to review and adjust the BOM. Though some users may desire this level of precision and granularity in their BOMs, most customers did not desire or need this level of matching. Therefore, individual file matching is no longer the default behavior and instead is an optional capability as of the Black Duck 2020.2.0 release.

This may cause some components to drop off your BOM, which may or may not be desired. Therefore, in the Black Duck 2020.2.0 release, Black Duck provides parameters in the scanning tools so that you can re-enable individual file matching. Refer to the Signature Scanner CLI and Black Duck Detect documentation for more information.

## When to use Black Duck Binary Analysis versus Black Duck

You may be confused as to when to use Black Duck Binary Analysis (BDBA) and when to use (standard) Black Duck source scanning.

BDBA is a security-focused application composition analysis scanner. It operates purely on provided binary images or compiled applications. There are a wide range of different binary file formats, CPU architectures, and programming languages/environments covered by BDBA. Refer to the BDBA FAQ located in the Black Duck Documentation Portal for more details.

Black Duck (without BDBA integrated) provides scanning capabilities focused on the development of applications or products based on analyzing source code. It can also recognize some binary files, but does not inspect binary files contents as does BDBA.

You may find it difficult to know which technology to use in your SCA solution. BDBA should be used to analyze:

- 3rd party software supplied in binary form, for example, in a software supply chain scenario
- Software in binary form where you do not have access to the source used to build it

Black Duck should be used when you have access to the source code or build environment used in the construction of software.

Sometimes the above use cases are intertwined. For instance, you have a software project based on the inclusion of one or more binary files from 3rd parties (free and open software (FOSS) and/or commercial). In this case, you might separate the analysis into two parts – the binaries are analyzed using BDBA and the source code is analyzed using Black Duck. Then merge the results to get a total BOM for your project.

Both products support analysis of docker images, although they approach the problem in different ways and can therefore produce different results. But the same guidance provided above applies. If you are building a docker image you may find it more effective to use Black Duck to analyze what you put onto a base docker image and then analyze the base image using BDBA. Or you can analyze the image using both products and merge the results to have a more complete picture of what is inside the image. On the other hand, if you are receiving, or using, images without any access to the source code used to build the image or to the code layered onto the image, BDBA may prove a better fit to analyze the image contents. Some experimentation may be required to determine the best fit for your needs.

Additional details can be found in the BDBA FAQ and product-specific guides for each product.

# Configuring automated scans

Setting up automated scans, for example, in a build server or CI/CD context, requires thinking about:

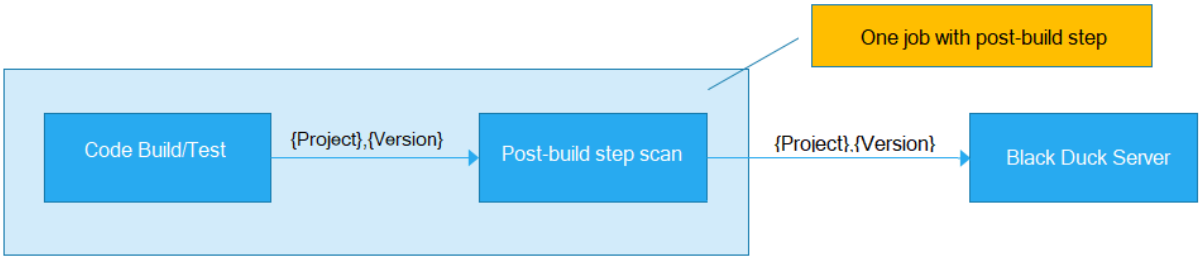
- Where/when in the build process to invoke the scan
- Whether to perform a full or rapid scan and if full scanning, whether the scan should be synchronous or synchronous
- Scan name(s), project version to map the scan (or scans) to, and versioning

## Where/when in the build process to invoke the scan

When using a build system such as Jenkins, scans should be run as a post-build step. Performing the scan as a post-build step ensures we can maximize the accuracy of the scan results by scanning the package manager files (if present), the resulting binaries, and the source code.

Another consideration when configuring automated scans is the size and complexity of what is being scanned. It might be advisable to split the scanning into multiple smaller scans if the project being scanned is particularly large (for example larger than 1 GB) or complex (for example, having more than 10,000 files), but this limit does not apply to package manager-only scanning.

For small projects, rapid scanning or package manager-only full scanning in a single post-build step is usually sufficient.

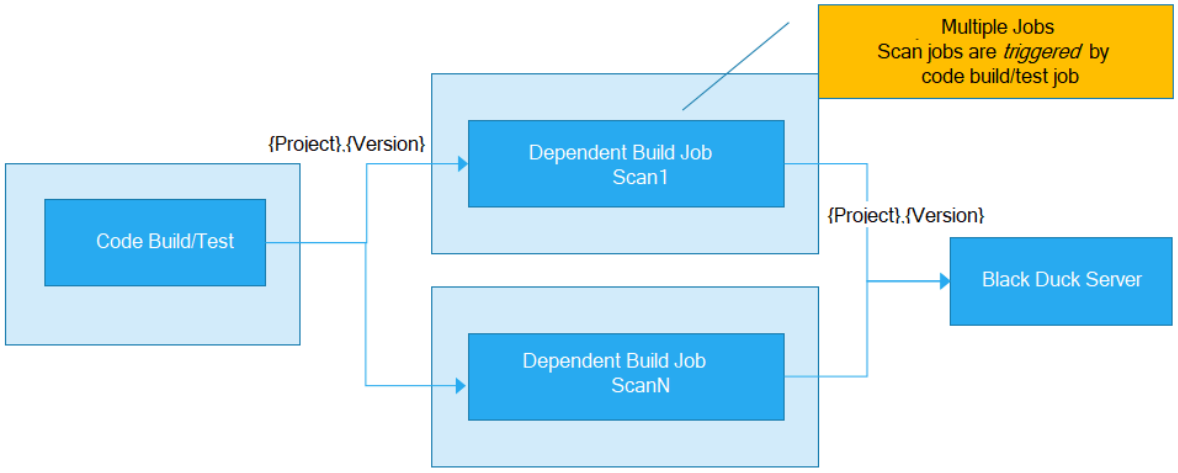



For large projects using the Signature Scanner, Black Duck recommends splitting up the scanning into multiple smaller scans that all get mapped to the same project version to aggregate the results. There are multiple ways to split the scanning, and most do not require any change to how the code is built.

For instance, suppose you have a large project that is spread across multiple folders. Instead of having one scan starting from the parent folder for the project, configure scans on each of the subfolders. Each scan is still mapped to the same project version and, as a result, overall scan time will be reduced, and the results will still be aggregated in Black Duck.

As shown, in Jenkins, this can be done by having multiple, downstream jobs scan the subfolders. Or, you could write a shell script that does multiple scans as a post-build step within the parent job.

**Note:** Alternatively, you can map each of the scans to a separate project version and build a project version hierarchy to aggregate results.



## Asynchronous versus synchronous full scans

When setting up the automated full scans, determine if the scans will be asynchronous (default) or synchronous. If the intent is to provide immediate feedback and “break the build” when policy violations occur, use a synchronous scan. However, if performance is more important than providing immediate feedback, then an asynchronous scan is the best choice.

## Scan names, project versions, and versioning for full scanning

We want to configure scans to generate the business results needed and to minimize having to delete or purge items from the database over time. The business results needed usually include:

- Giving developers early warning that the open source they are using has risks associated with it, so they can choose appropriate alternatives to avoid those risks
- Retaining knowledge of software that has been distributed (for example, within a product, or as part of a service run in production)
- Providing the organization with an overall view of the open source in use and the risks associated with it

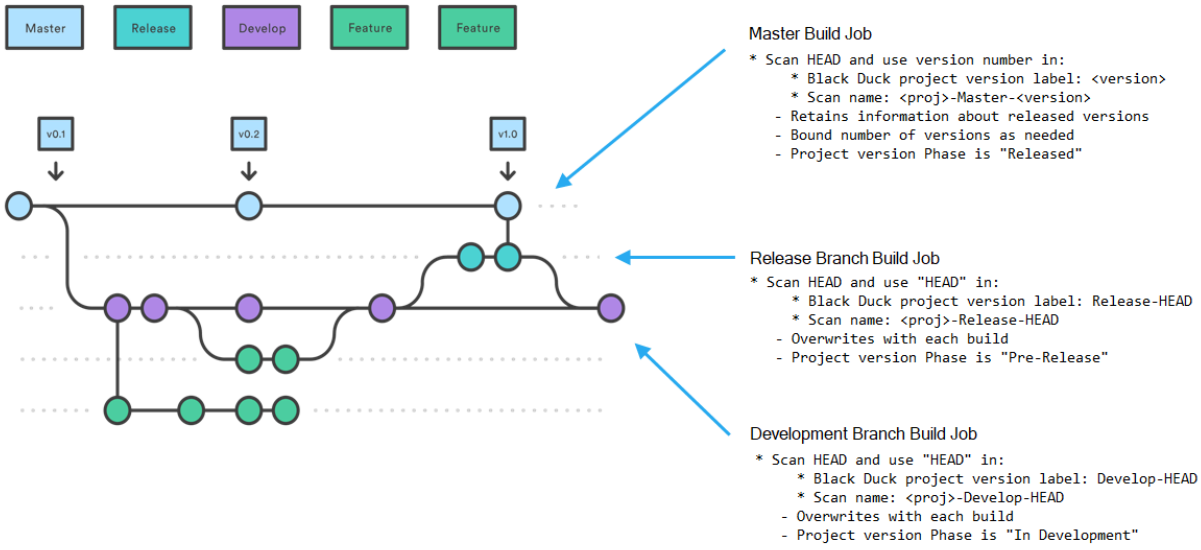
We want to avoid setting up full scans in a way that leads to unbounded growth of data since that clutters the system and can cause system performance to degrade very quickly. And we want to avoid the need to purge or delete items since that is often neglected or forgotten and can be time consuming in any case.

How do we do this?

## Projects using multiple branches

In projects with multiple branches there is invariably one branch that is used to release/distribute software from, and the others are used for development/testing leading up to a release (also known as distribution). We want to configure scanning such that we retain the information corresponding to what is distributed (released). If scans are configured on other (non-release) branches, for example, to give early warning to developers, those scans should be rapid scans or, if using full scans, always overwrite the previous scan results. Setting up the scanning this way allows us to achieve our business goals while minimizing scan/version proliferation.

The best way to describe the method is by looking at the branches in a Gitflow model. In the Gitflow model, software is only distributed (released) from the Master branch and each version is given a label which comes from source control (for example, GIT). Other branches, such as Release or feature branches, are used for intermediate builds and do not get distributed.




This diagram depicts configuring three types of build/scan jobs that occur whenever a commit happens on their respective branches. The next sections provide details on how to configure scans for each branch.

## Rapid scanning a working branch

Developers' working branches should be scanned using the rapid scanning mode, and where possible, a project name, release name and/or branch name where applicable should be provided using the rapid scanning. No results or a BOM will be kept on the server. Thousands of such rapid scans per day can easily be run against a single server. Rapid scans give a quick check of policy violations for the projects open source components. Thus, security or licensing policy violations need never enter the code base.

## Full scanning the Release branch

The scan on the Release branch uses the name "HEAD" for the project version and scan names to ensure that the latest scan and BOM results are always overwritten. As the diagram shows, we configure the scan of this branch to use the scan name <proj>-Release-HEAD where <proj> represents the project name. The other sections of the scan name are fixed so that each time a new scan occurs it overwrites the previous scan. The project version label is set to "Release-HEAD" and the project version phase is set to **Pre-Release** to indicate that this scan is for code that is about to be released.

This gives development and security management an early warning about components they have chosen which do not conform to policy or perhaps include high risk security vulnerabilities.

We do not care about retaining history, and it would be wrong to keep prior versions because it will clog your system and does not provide any business benefit.

## Full scanning the Master branch

In Gitflow, all releases happen off the Master branch, and we want to retain knowledge of released versions because those are the ones that run in production or are distributed to customers. We therefore use a version number, usually pulled from a file in the source code, and use it as part of the project version and scan names to ensure that the scan and BOM results are retained for all released versions.

By maintaining all the released versions, we ensure that the Black Duck system can notify you about future vulnerabilities that affect components used in those released versions.

By only creating full scans for controlled release versions we keep the number of versions retained in the system to a minimum.

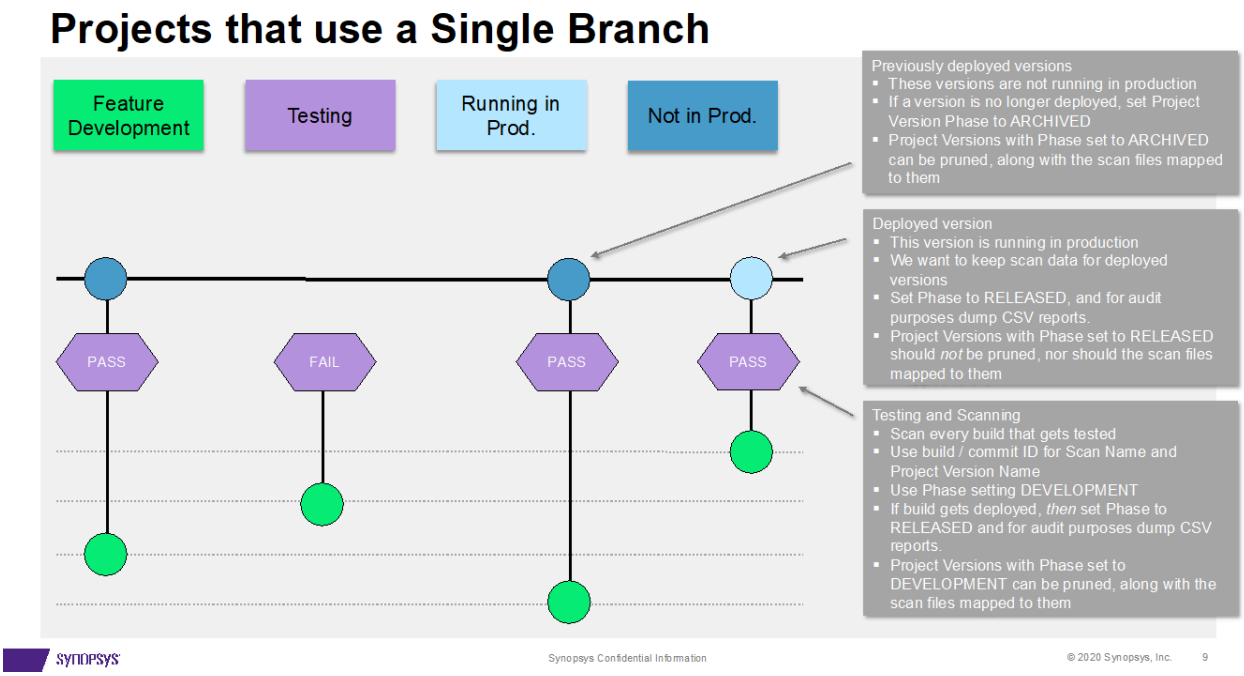
This project version has the **Released** phase.

The following table shows the project version and scan location names if your project is named MY-SAMPLE-PROJECT:

| Branch | Scan/Code Location Name (--detect.code.location.name property) | Project Version Name (--detect.project.version.name property) |
|---|---|---|
| Working Branch | $(branch_name) | $(version_label_from_source_code_if_available |
| Release | MY-SAMPLE-PROJECT-Release-HEAD | Release-HEAD |
| Master | MY-SAMPLE-PROJECT-$(version_label_from_source_code) | $(version_label_from_source_code) |

## Projects using a single branch

Customers developing web- or micro-services usually do their development on a single branch.




In these projects a stream of commits is tested and optionally pushed to production based on the test outcomes. Each version must be scanned at the time of the commit, but which version makes it to production is not known until sometime afterward. In this case, Black Duck recommends:

1. Using a build or commit ID in the scan and version name to distinguish versions and scans.
2. Use phase setting DEVELOPMENT for the initial scan.
3. When tests indicate a version will be pushed to production (in other words, released), automate marking the phase of that version as RELEASED. Mark all other versions, previously marked as RELEASED, as Archived or DEVELOPMENT so they can be removed from the system based on age.

4. Automate deleting old versions, and scans, whose phase is non-RELEASED to retain only the (one) RELEASED version.

For audit purposes, save the `.csv` reports for all RELEASED versions off-line.

## Scanning docker images

Another common scanning use case is to analyze a docker image using the Black Duck Detect Docker Inspector module. The simplest and sometimes most effective approach is to scan the whole image which results in a combined list of components, and vulnerabilities, for the entire image.

Often, customers want to separate issues pertaining to the base image(s) from the application code they load into it. In that case, break down the analysis of a docker image into two parts:

1. The base image (that is, the image named in the FROM statement), and
2. What you put on top of the base image, that is, your application

The way this is done is:

1. Analyze the base image using Docker Inspector, mapping the results to its own project-version.
2. Now, using the docker image that combines the base and your application code, obtain the top layer ID (the layer above which your application code resides).
3. Use the docker inspector with the **--docker.platform.top.layer.id** property, analyze the combined image (base + application) and provide the top layer ID from step 2. Map the results to a separate project version which will contain only the parts loaded on top of the base image.

## Signature scanning very large projects

If you have a very large project (for example, a 4.5 GB monolithic project with 500+ files) and need to use the Signature Scanner, your initial instinct may be to scan the entire project in one full scan. However, this is the incorrect approach as a single scan of a very large project can stress your Black Duck system (you may need to add additional RAM and CPUs to process the scan). If there are any changes — even if only a small portion of the code has changed — you will need to rescan the entire project each time and look at the results for the entire project. Additionally, you lose the ability to have a flexible policy management system (for example if you want different subcomponents managed differently), which would not be possible with a single scan.

The correct way to manage signature full scanning for a large project is to divide the project into multiple scans by using multiple dependent jobs. These multiple scans can then become subprojects which you aggregate into one parent project. This scanning strategy has these benefits:

- Only need to full scan and inspect what has changed.
- Can still report on the entire project.
- Provides a flexible policy management system:
  - Different sub-components can be managed differently.
  - The parent can still be treated as a single entity.

**Note:** The size limit for an individual scan is 5 GB. An error message appears if you exceed this limit. Contact Customer Support if you receive this message.

## Handling reusable modules (libraries)

Suppose you want a BOM for a library which is contained in one folder to be combined with an application that sits in another folder. This section describes the two methods you can use to accomplish this by:

- Aggregating multiple full scans into the same project version.
- Creating a BOM hierarchy.

## Aggregating multiple full signature scans

To aggregate the results of multiple scans, point those scans to the same project and version. For example, in the example of a library that sits in one folder and an application (that uses the library) in a separate folder:

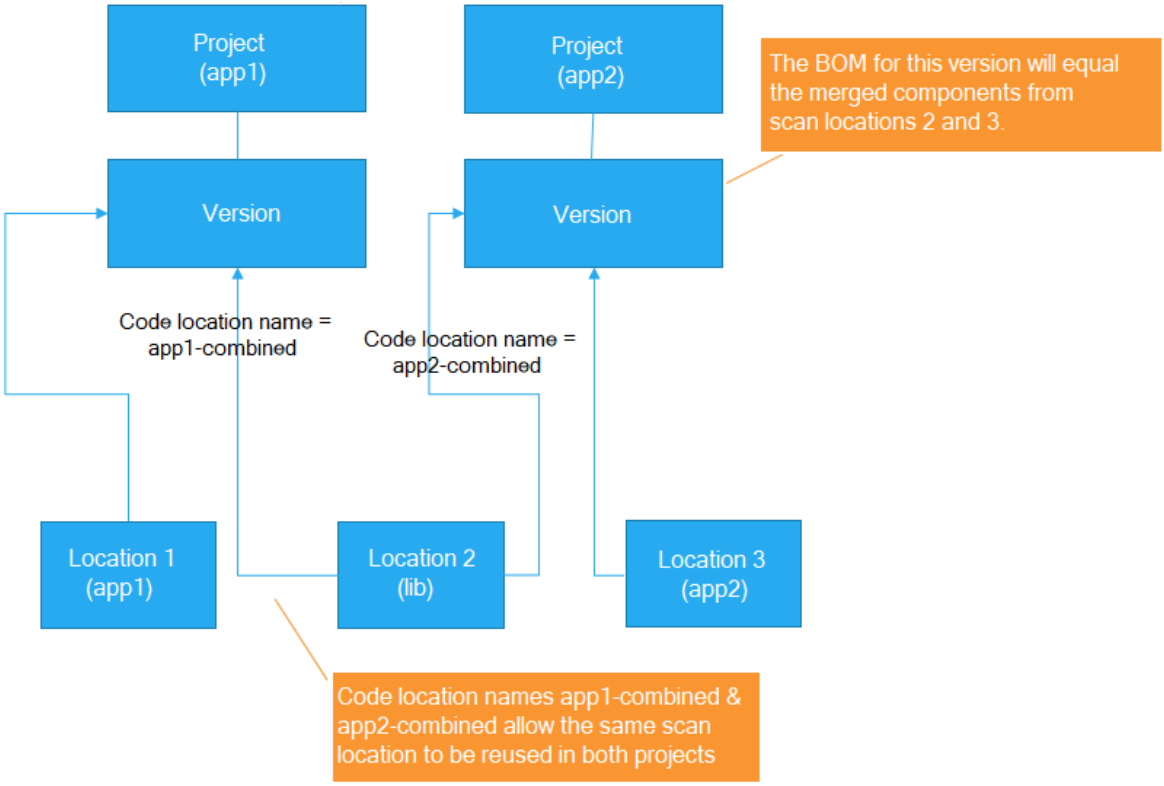
1. Scan the library folder and map the scan to a project (for example, my-application) and version (for example, 1.0).
2. Scan the application folder and map the scan to the same project (my-application) and version (1.0).

Note:

- The resulting aggregate BOM represents both the library and the application.

  Note that the **Component** tab for the project version displays a "flat" view of components — all components found during the full scans – regardless of the directory were the component was found — are listed at the same level on the **Component** tab.
- Use the **--detect.code.location.name** property to avoid scan name conflicts. This allows reuse of the same physical location into multiple projects. This is useful if the library is used by more than one application and you want the contents of the library reflected in all the applications that use it. In that case, scan the same library folder multiple times giving each scan a unique name and map the scan to the application that uses it.
- You must rescan to update the combined BOMs.

For example:

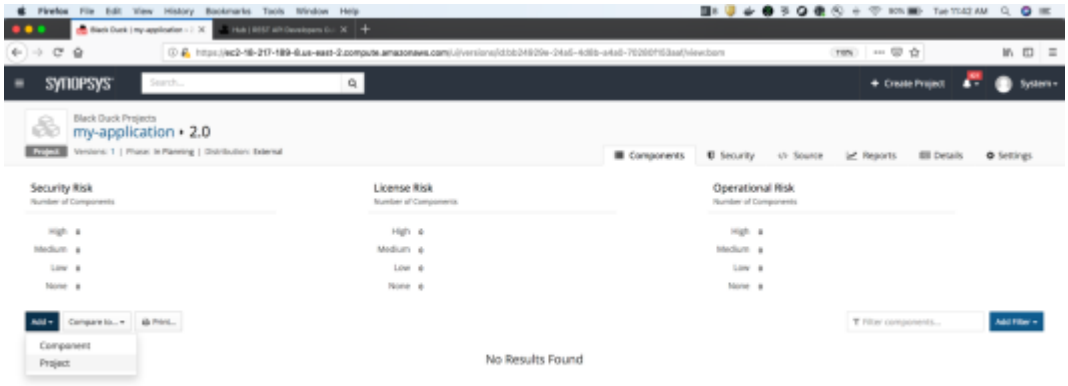



## Using a project hierarchy

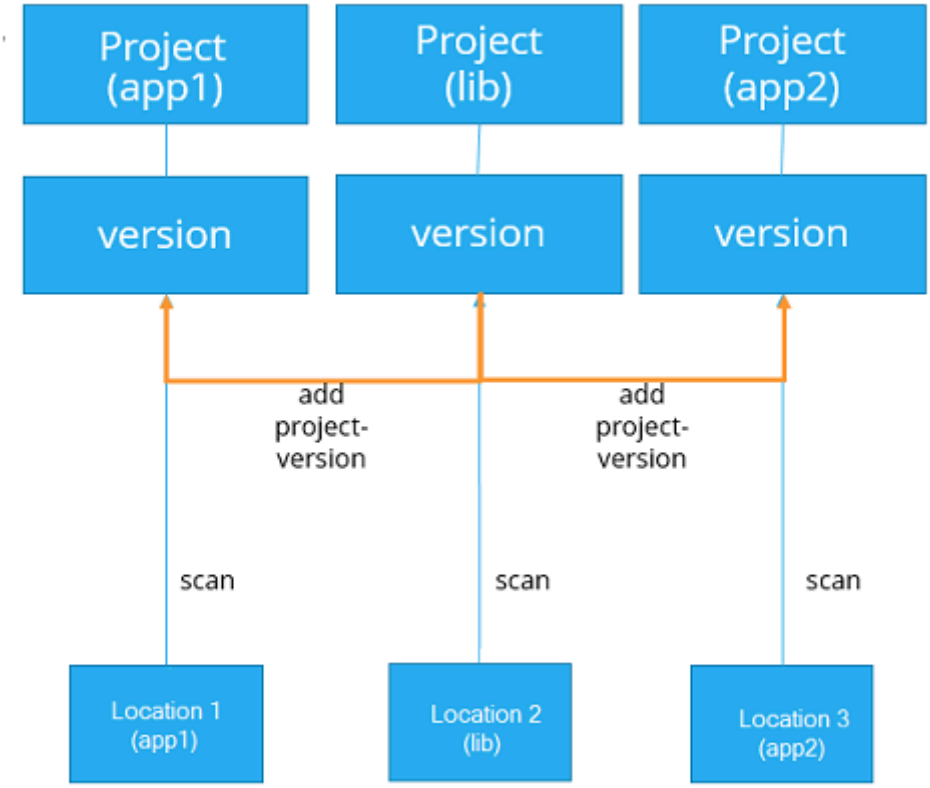
In this option, use a different project and version name in Black Duck Detect for each full scan. In the parent BOM, add the library module project version as a subproject. In this scenario, subsequent full scans/ changes to the subproject are immediately reflected in the parent BOM.

So, in our example of a library in one folder and an application in a separate folder, do the following:

1. Scan the library folder and map it to its own project (for example, my-library) and version (for example, 1.0).
2. Scan the application folder and map it to a separate project (for example, my-application) and version (for example, 2.0).
3. In the Black Duck UI, navigate to the **Component** tab for application project (my-application) version (2.0).
4. Add the library's project (my-library) to the BOM of the application project (my-application, version 2.0): click *Add* and select **Project** from the list. The UI prompts you for the version (1.0) and after you confirm the addition, the component list from the library project version will be included in the application's BOM:




The diagram below depicts the case where a library is used by two applications. Project (lib) is added as a subproject to the BOM of Project (app1) BOM and/or Project (app2).




Using a project hierarchy in this way isolates the full scans of the library (child project) from the applications (parent projects) that use it. It also encapsulates the BOM for the library from the projects that use it, so you can view and interact with the library and its contents separate from the application BOM which simply includes the library's results.

## Scanning when network connectivity is an issue

If network connectivity is an issue, use a store and forward strategy:

1. Use the Black Duck Detect **--blackduck.hub.offline.mode=true** property. This disables all communication with Black Duck but still generates metadata.
2. Forward and then upload to Black Duck separately.

# Poor scanning techniques

## Including the commit ID or build ID in the Black Duck version or scan names

Some customers have been reporting poor performance of their Black Duck server. There are several ways that this can happen, but for many of these customers their Black Duck server/database was filled with hundreds or even thousands of spurious versions and full scans because they used either a commit ID or build ID in the project version name or scan name when they configured their scans for their automated builds. In some cases, without realizing it, they had configured a system of continually creating new versions and full scans with no thought to purging old ones or had no effective automated solution for keeping the system from clogging up.

Once their system got to the point of having all those hundreds or in some case thousands of unwanted versions or scans, cleanup becomes a huge issue since deletion is slow in an overloaded system and deleting scans/versions is a scary prospect when you are perhaps uncertain what is important to keep and what you can discard. It is much easier if you never get to this point in the first place.

This is more than a performance degradation issue. This also makes the resulting project version BOM inaccurate since the builds occur over time and the resulting BOM is the aggregation of all the full scans.

## Keeping a history of versions on a development branch when full scanning

Some users say they want to keep a history of the versions on a development branch so that they can determine when the development team introduced an issue.

However, Black Duck was not designed to be a data warehouse, keeping all information forever. If you are trying to use the Black Duck system in this manner, your server/database will be overwhelmed with too much data. Performance will begin to severely degrade, and the system will become unusable.

If your goal is to understand forensically what has happened over time, the appropriate way to approach this is to export the scan results from the system (for example, using the Black Duck reporting database or REST APIs) to a data lake or data warehouse that is designed to accumulate large amounts of data and supports the slicing/dicing of the data. That capability is outside of the scope of what the Black Duck server was designed to do.

## Avoid Parallel Scanning of the Same Code Location

Parallel scanning of the same code location using the same `scan_name` (code.location.name) can lead to conflicts and unexpected results due to architectural limitations. To ensure accurate and reliable scan outcomes, adhere to the following best practices:

- **Queue or Cancel Overlapping Scans:** Avoid triggering multiple scans with the same `scan_name` simultaneously. Instead, queue scans or cancel overlapping ones to prevent resource conflicts and race conditions.
- **Limit Scan Containers:** Use a single scan container per deployment to eliminate conflicts. Be aware, however, that this approach may skip scans that are already in progress.

- **Understand Architectural Constraints:** Scan containers are not designed for parallelism. Running multiple scan containers on the same code location splits resources and causes race conditions, leading to incorrect or incomplete results.
- **Monitor Scan Status:** Utilize UI and API tools to monitor scan status and avoid triggering parallel scans unintentionally. Improved scan visibility tools are being developed to help customers manage scan workflows effectively.

**Note:** Future architectural improvements, such as scan ID protocols and removing synchronous locks, are planned to enhance scan handling and coordination. Until these changes are implemented, customers should avoid frequent parallel scans of the same code location and follow these updated best practices.

# 2. Rapid Scan Overview

Rapid Scan SCA is a new way of scanning within Black Duck. This mode is designed to be as fast as possible and support developer workflows without persisting any data on Black Duck.

It is enabled by adding `--detect.blackduck.scan.mode=RAPID` to a run of detect.

Unlike existing scans, no data is retained on Black Duck and all scans are done transiently. These scans are primarily intended to be fast. The results are saved to a json file in the Scan Output directory, where name and version are the project's name and version.

Rapid Scan relies on Black Duck policies to produce SCA acceptance testing results which can be integrated into developer pipelines. Rapid Scan only reports components that violates policies. If no policies are violated or there are no defined policies, then no components are returned.

By default, rapid scan will fail with `FAILURE_POLICY_VIOLATION` if any component violate polices with a CRITICAL or BLOCKER severity

The results are also printed in the logs, looks like:

```
2021-06-18 11:38:18 EDT INFO [main] --- Rapid Scan Result: (for more detail look in
the log for Rapid Scan Result Details)
2021-06-18 11:38:18 EDT INFO  [main] ---
2021-06-18 11:38:18 EDT INFO [main] ---        Policy Errors = 0
2021-06-18 11:38:18 EDT INFO [main] ---        Policy Warnings = 99
2021-06-18 11:38:18 EDT INFO [main] ---        Security Errors = 7
2021-06-18 11:38:18 EDT INFO [main] ---        Security Warnings = 0
2021-06-18 11:38:18 EDT INFO [main] ---        License Errors = 0
2021-06-18 11:38:18 EDT INFO [main] ---        License Warnings = 0
2021-06-18 11:38:18 EDT INFO [main] ---
2021-06-18 11:38:18 EDT INFO [main] ---        Policies Violated:
2021-06-18 11:38:18 EDT INFO [main] ---            Security Vulnerabilities Great
Than or Equal to High
2021-06-18 11:38:18 EDT INFO [main] ---            Warn on Low Security
Vulnerabilities
2021-06-18 11:38:18 EDT INFO [main] ---            Warn on Medium Security
Vulnerabilities
2021-06-18 11:38:18 EDT INFO [main] ---
2021-06-18 11:38:18 EDT INFO [main] ---        Components with Policy Violations:
2021-06-18 11:38:18 EDT INFO [main] ---            Apache PDFBox 2.0.12 (maven:org.
apache. pdfbox: pdfbox:2.0.12)
2021-06-18 11:38:18 EDT INFO [main] ---            node-ini 1.3.5 (npmjs:ini/1.3.5)
2021-06-18 11:38:18 EDT INFO [main] ---
2021-06-18 11:38:18 EDT INFO  [main] ---        Components with Policy Violation
```

enabled by adding `--detect.blackduck.scan.mode=RAPID` to a run of detect.

Rapid scanning will check for violations based on Component Conditions, Vulnerability Conditions, and License conditions and a Json file to summarize and provide more details generated.

### Rapid Scanning Best Practices

Rapid mode is primarily meant for developers or devops processes used in the development stage. The intention is to provide development teams with quick access to high fidelity Black Duck results without being a bottleneck in development speed.

Black Duck Rapid Scan enables developers to get Black Duck results extremely quickly, and it supports thousands of scans per hour. Rapid Scan is focused on package managers and component security. It works in together with Black Duck Policy Rules, so developers can quickly see if any policy violations will be introduced when checking-in their work. Companies can also implement this scan model as part of their CI / CD pipeline.

### Use case for Developers

Unlike existing Black Duck scans, no data is persisted on Black Duck once scans are completed. These scans are primarily intended to be fast and integrated into developer workflows. The best practice is to use rapid scan mode to quickly fail a build if any policy violations are introduced in the code in the development stage. The results show the developer which policies have been violated, and which declared OSS components are in violation. Rapid Scan can also fail a build, for Blocker or Critical policy violations. Rapid scanning will check for violations based on Component Conditions, Vulnerability Conditions, and License conditions.

### Rapid Scan specific policies

With Rapid Scan specific policies in Black Duck, a rapid scan can be configured to fail a build on specific policy violation conditions catered to Rapid Scan only. For example, if an organization wants to ensure that no code with component vulnerability score of 5 and above should be merged in the development stage, they can create a Rapid scan specific policy and builds will fail in Detect when triggered.

### Rapid Scan policy overrides

Policy overrides for a rapid scan can be provided using a scan custom config file, provided in a file named `.bd-rapid-scan.yaml` in the source directory. The file name must match exactly. Please see the Configuration section of Detect's Rapid Scan page for an example of the `bd-rapid-scan.yaml` file.

This provides additional flexibility for organizations on their usage and implementation, or relaxation of build break rules based on rapid scan policy violations.

### Rapid scan differential feature: Only show NEW violations since the last full scan

With this functionality, organizations can now rapidly find out if new violations have been introduced since the branch was scanned with the full scan, i.e. compare against a project version on Hub. They can fail a build only if there are new issues introduced by a branch.

If the policy violation was already known and visible in the project version's component page (active or overridden), it will not be considered in the Rapid Scan results. Only new violations found in the scanned project are returned.

By using the `X-BD-RAPID-SCAN-MODE` header or detect option `detect.blackduck.rapid.compare.mode` and providing it one of the values below, you can change the Rapid Scan mode.

Set the compare mode of rapid scan:

- `ALL`: The default operation. It will evaluate policy rules that are `RAPID` or (`RAPID` and `FULL`). When the header is absent, this is the default behaviour.
- `BOM_COMPARE`: Will evaluate policy rules like the `ALL` option, but will now evaluate differently based on the type of policy rule modes. When the policy rule is (`RAPID` and `FULL`) it will behave like `BOM_COMPARE_STRICT` but if the policy rule is only (`RAPID`) it will evaluate the result of the rapid scan against the policy ignoring results in the BOM.
- `BOM_COMPARE_STRICT`: Will only evaluate policy rules that are (`RAPID` and `FULL`). Policy violations are compared to the existing project version BOM. If the policy violation was already known and visible in

the BOM (active or overridden) it is not part of the rapid scan positive result, it will still be part of the full result following existing restrictions.

In order to run either of the `BOM_COMPARE` modes there must be an existing project version in HUB.

**When NOT to run Rapid scan?**

Rapid scan does not create a Project or Version on Black Duck, so it is not meant to run to generate a BOM. It cannot create a Risk or Notices report.

**Interactive Tutorial**

Explore this interactive tutorial to try out Black Duck's Rapid Scan.

# 3. Troubleshooting scanning issues

This chapter describes some scanning problems and provides solutions.

## Accidental full scan proliferation by folder paths which include build or commit ID

Suppose your build server performs a full scan of the same project repeatedly and the full scan is mapped to the same project version each time. The build server creates a workspace folder which includes a build or commit ID in the path. If a scan name was not provided, Black Duck Detect creates a scan name which is derived from the full folder path. Because that path includes an element that changes with each build, the result is a growing list of scans that all are mapped to the same project version. The resulting BOM for the project version will be the sum of all components identified in all the scans, that is, across all the builds.

### Solution

1. If the goal is to provide quick feedback to developers and the results do not need to be persisted, switch to a rapid scan by using the following property:, **--detect.blackduck.scan.mode=RAPID**. Doing this will eliminate the problem because rapid scans only show the results for the code scanned, and it is not possible to integrate results from previous scans.
2. If the results should be persisted, supply a scan name using the **--detect.code.location.name** property so that each build over-writes the prior scan results. By over-writing the previous scans, the issue is eliminated, and we can ensure that the BOM is accurate. This also avoids accumulating lots of old scan data.

Click here for more information on Black Duck Detect properties.

## Accidental full scan proliferation by a build server farm

If you have a build server farm creating your builds, then each time a build occurs there is potentially a new host (up to the number of hosts in the farm). If you do not supply your own scan name, Black Duck Detect derives one that includes the hostname and therefore, you will get a different scan for each host. All full scans will be mapped to the same project version in Black Duck and the result is multiple redundant full scans mapping to the same project version.

### Solution

1. If the goal is to provide quick feedback to developers and the results do not need to be persisted, switch to a rapid scan by using the following property: **--detect.blackduck.scan.mode=RAPID**. Doing this will eliminate the problem because rapid scans only show the results for the code scanned, and it is not possible to integrate results from previous scans.
2. If the results should be persisted, supply a scan name using the **--detect.code.location.name** property so that each build over-writes the prior scan results. By over-writing the previous scans, we eliminate the issue and ensure that the BOM is accurate. This also avoids accumulating lots of old scan data.

Click here for more information on Black Duck Detect properties.

# 4. Frequently Recommended Black Duck Detect Properties

Below are some of the more frequently used Black Duck Detect properties and their use.

| Task | Property |
|---|---|
| Check for policy violations | • **--detect.policy.check.fail.on.severities**<br>A comma-separated list of policy violation severities that will fail Black Duck Detect. If this is not set, Black Duck Detect will not fail due to policy violations for full scans.<br>• **--detect.timeout**<br>When using the policy check property above, you may need to increase the timeout for larger, more complex projects. |
| Perform a Rapid Scan | • **--detect.blackduck.scan.mode=RAPID**<br>Use this property to run a package manager only, synchronous scan, returning scan results to the command line, without creating a BOM or saving results in Black Duck. Defaults to false. |
| Disable signature (also known as file system) scanning and rely on package manager scanning exclusively | • **--detect.tools=DETECTOR**<br>Runs the Detector tool only. |
| Include and exclude options to tune what gets analyzed by the Signature Scanner | • **--detect.blackduck.signature.scanner.exclusion.patterns**<br>Enables you to exclude the folder matching the absolute path from the scanning target folder.<br>• **--detect.blackduck.signature.scanner.exclusion.name.patterns**<br>Enables you to provide folder patterns to exclude. Black Duck Detect will search all folders inside the scanning target and then exclude those matching the supplied patterns.<br>• **--detect.blackduck.signature.scanner.paths**<br>Enables you to specify that these paths and only these paths will be scanned for full scanning. |
| Enable Correlated Scanning | • **--detect.blackduck.correlated.scanning.enabled**<br>When enabled, Black Duck Detect activates the Black Duck SCA correlated scanning capability to enhance match accuracy. The correlated scanning capability must be present and enabled in your Black Duck SCA server before you enable the correlated scanning feature in Black Duck Detect. |