Synopsys Detect 8.10.0

Contents

1

Introduction to Detect

Synopsys Detect is Black Duck's intelligent scan client that scans code bases in your projects and folders to perform compositional analysis. Synopsys Detect sends scan results to Black Duck, which generates risk analysis when identifying open-source components, licenses, and security vulnerabilities.

Synopsys Detect can be used in both connected and air gap modes.

Synopsys Detect has the following characteristics.

- Synopsys Detect integrates with development tools used throughout the SDLC (software development life cycle) and automatically detects resources to optimize its scan methodology.
- Synopsys Detect provides scanning capabilities to Black Duck to help identify open-source components, licenses, and security vulnerabilities. This is achieved through a variety of detection methods such as package manager inspection, file system based signature scanning of source directories and files, Docker image inspection, and binary analysis.
- Synopsys Detect provides the source of information for Black Duck to analyze open-source components and find vulnerabilities in open-source components and containers. Using this type of analysis, you can minimize security, compliance, and code quality risks; you can monitor for new vulnerabilities throughout your development cycle, and you can set and enforce open-source use and security policies.
- Runs on Windows, Linux, and macOS. It is available through GitHub, under the permissive Apache License, Version 2.0 and does not require pre-installation or configuration.
- Supports scanning Docker images by identifying open-source libraries and code within the images, using both signature scanning and the package manager analysis techniques.

Synopsys Detect functionality consolidation.

Synopsys Detect consolidates the functionality of Black Duck, package managers, and continuous integration plugin tools to perform the following tasks:

- Discover open-source components in your code.
- Map components to known security vulnerabilities.
- Identify license compliance and component quality risks.
- Set and enforce open-source use and security policies.

- Integrate open-source management into your DevOps environment.
- Monitor and alert users when new security threats are reported.
- Calculate security vulnerability risk in your code.
- Produce reports of the open-source analysis findings.

How Synopsys Detect functions.

When looking at vulnerabilities in open source and third-party software, Synopsys Detect performs the following basic steps:

- Uses the project's package manager to derive the hierarchy of dependencies known to that package manager. For example, on a Maven project, Synopsys Detect executes an mvn dependency:tree command and derives dependency information from the output.
- Runs the Black Duck signature scanner on the project. This might identify additional dependencies not known to the package manager (for example, a .jar file copied into the project directory).
- Uploads both sets of results (dependency details) to Black Duck creating the project/version if it does not already exist. Black Duck uses the uploaded dependency information to build the Bill Of Materials (BOM) for the project/version.
- You can view the output and analysis results in Black Duck.

2

Release Notes

Topics:

Current Release notes

Version 8.10.0

Changed features

- Leading and trailing spaces specified within quotes for `detect.project.name` or `detect.project.version.name` properties will now be trimmed.

Resolved issues

- (IDETECT-3657) Resolved an issue where Intelligent Scans would fail if a project or version name included non-ASCII characters.
- (IDETECT-3776) Resolved an issue with not detecting certain components in `go.mod` files as transitive dependencies when marked with `// indirect`, by improving identification of direct and indirect dependencies.
- (IDETECT-3817) Improved handling of large inspection results to prevent OutOfMemory exceptions and optimize memory usage.
- (IDETECT-3888) Improved the runtime performance of PIP Inspector for aws-cdk dependency cases by passing the package history list by reference instead of value.
- (IDETECT-3867) Resolved a lack of support for properties set in SPRING_APPLICATION_JSON environment variable for configuring Synopsys Detect when the Self Update feature is utilized.

Dependency updates

- Upgraded Spring Boot to version 2.7.12 to resolve high severity *CVE-2023-20883*

- Upgraded SnakeYAML to version 2.0 for Synopsys Detect air gap package to resolve critical severity *CVE-2022-1471*
- Upgraded Jackson Databind to version 2.15.0 for Synopsys Detect air gap package to resolve high severity *CVE-2022-42003* and *CVE-2022-42004*
- Upgraded Project Inspector to version 2021.9.9

Version 8.9.0

New features

- Synopsys Detect Self Update feature will allow customers who choose to enable Centralized Synopsys Detect Version Management in Black Duck to automate the update of Synopsys Detect across their pipelines. The Self Update feature will call the '/api/tools/detect' API to check for the existence of a mapped Synopsys Detect version in Black Duck. If a version has been mapped, the API will redirect the request to download the specified version and the current execution of Synopsys Detect will invoke it to execute the requested scan. If no mapping exists, the current version of Synopsys Detect matches the mapped version in Black Duck, or if there is any issue during the execution of the Self Update feature, then Synopsys Detect will continue with the currently deployed version to execute the scan.
  - Centralized Synopsys Detect Version Management feature support in Black Duck is available from Black Duck version 2023.4.0 onwards.
  - See *Version Management* for more details.

Changed features

- Release notes are now broken into sections covering the current, supported, and unsupported Synopsys Detect releases.
- npm 6 has reached end of life and is being deprecated. Support for npm 6 will be removed in Synopsys Detect 9.

Resolved issues

- (IDETECT-3613) Resolved an issue where running a scan with `detect.maven.build.command=-Dverbose` caused a KB mismatch issue for omitted transitive dependencies.

Dependency updates

- Upgraded SnakeYAML to version 2.0 to resolve critical severity *CVE-2022-1471*
- Upgraded Jackson Dataformat YAML to version 2.15.0 to resolve critical severity *CVE-2022-1471*
- Upgraded Spring Boot to version 2.7.11 to resolve high severity *CVE-2023-20873*

Version 8.8.0

New features

- New Binary Stateless and Container Stateless Scans have been added to Synopsys Detect. These scans require the new detect.scaaas.scan.path property to be set to either a binary file or a compressed Docker image. See the *Stateless Scans page* for further details.

  Attention: A Black Duck Binary Analysis (BDBA) license is required to execute these scan types.

Changed features

- Evicted dependencies in Simple Build Tool(SBT) projects will no longer be included in the Bill of Materials(BoM) generated during the scan.
- Introduced an optional flag to allow a space-separated list of global options to pass to all invocations of Project Inspector. Specify the `--detect.project.inspector.global.arguments` flag in the command, followed by other global flags if needed for pass through to Project Inspector. See *project-inspector properties for further details*.
- The maximum polling interval threshold is now dynamic when Synopsys Detect polls Black Duck for results. This dynamic threshold is dependent upon, and optimized for, the specific scan size. (The maximum polling threshold was formerly a fixed 60-second value.)

Resolved issues

- (IDETECT-3111) When scanning SBT projects, "Evicted" dependencies are excluded from the resulting BOM.
- (IDETECT-3685) Gracefully handled use case when a Podfile.lock file has no PODS or dependencies in the generated dependency graph.

- (IDETECT-3738) Repositioned the global flags for inclusion before sub-commands for Project Inspector invocation.

Version 8.7.0

New features

- The accuracy of dependency determination, HIGH or LOW, of any detectors run during a scan will now be recorded in the status.json file.
- STATELESS/RAPID scans, when run against Black Duck 2023.1.2 or later, will provide upgrade guidance for mitigation of vulnerabilities in transitive dependencies.

Changed features

- Addition of command line help option, -hyaml, to generate a template configuration file.
- Synopsys Detect's generated air gap zip is uploaded to Artifactory under the name "synopsys-detect--air-gap-no-docker.zip". Older naming patterns for this file are no longer supported.
- Failures in detectors will now be reported in the console output using the ERROR logging level. The ERROR logging is also used if there are errors in the overall status.
-

Resolved issues

- (IDETECT-3661) Synopsys Detect will fail and echo the error received from Black Duck, if a problem occurs during the initiation of a Stateless Signature Scan.
- (IDETECT-3623) Synopsys Detect will now fail with exit code 3, FAILURE_POLICY_VIOLATION, if Black Duck reports any violated policies during scans.
- (IDETECT-3630) Notices and risk report PDFs now appropriately contain the supplied project and version name when characters from non-English alphabets are used.
- (IDETECT-3654) As of version 8.0.0 of Synopsys Detect, Cargo project dependency graphs stopped being post-processed. Previously, attempts to define parent relationships for dependencies when the Cargo.lock file is a flat list resulted in marking any dependencies with a parent relationship as Transitive. This meant a dependency, which if Direct, may appear as Transitive in Black Duck if it is also a dependency of another component. BOMs created with 8.0.0 or later, no longer assume any relationships and all dependencies are DIRECT.

Version 8.6.0

Changed features

- Package Manager and Signature Scans will now query Black Duck directly when using the detect.wait.for.results property. This expedites scanning by allowing Synopsys Detect to determine if results are ready, rather than waiting for a notification from Black Duck. Note: this feature requires Black Duck 2023.1.1 or later.

Resolved issues

- (IDETECT-3627) When waiting for results, Signature Scans will now wait for all scans that the Signature Scan could invoke, such as Snippet and String Search scans. Previously, only the Signature Scan itself was checked for completion. Note: this improvement requires Black Duck 2023.1.2 or later.

Dependency updates

- Upgraded Apache Commons Text to version 1.10.0.
- Upgraded Docker Inspector to version 10.0.1.

Version 8.5.0

New features

- Added property blackduck.offline.mode.force.bdio which when set to true will force Synopsys Detect used in offline mode to create a BDIO even if no code locations were identified.

Changed features

- The .yarn directory will now be ignored by default when determining which detectors are applicable to a project.
- An exit code of 2, representing FAILURE_TIMEOUT, will be returned when STATELESS scans do not report status in a timely fashion. The timeout can be controlled using the detect.timeout property.

Version 8.4.0

Changed features

- The flag value EPHEMERAL has been deprecated in favor of the value STATELESS. See the *Stateless Scans page* for further details.

Resolved issues

- (IDETECT-3384) Changed Warning message "No dependency found" in Lerna projects to Debug level.

Version 8.3.0

New features

- Added support for Reduced Persistence Signature Scanning. This feature allows users to specify if unmatched files should be persisted or discarded. Not storing data for unmatched files decreases scan time and database size. Note: this feature requires Black Duck 2022.10.0 or later.

Resolved issues

- (IDETECT-3285) go.mod file "// indirects" matching as Direct Dependencies. Additional information for the go project is obtained in order to definitively establish direct module dependencies and then establish which module dependencies are transitive.
- (IDETECT-3228) Resolved an issue that caused certain Maven dependency tree formats to not be parsed.

Version 8.2.0

New features

- Ephemeral Scan, or Ephemeral Scan Mode, is a new way of running Synopsys Detect with Black Duck. This mode is designed to be as fast as possible and does not persist any data on Black Duck. See the *Ephemeral Scans page* for further details.

- The output for Rapid and the new Ephemeral Scan Modes will now include upgrade guidance for security errors and warnings.

Version 8.1.1

Resolved issues

- (IDETECT-3509) Corrected the version of the NuGet Inspector built into the air gap zip files (from 1.0.1 to 1.0.2).

Version 8.1.0

New features

- Added support for Bazel project dependencies specified via a github released artifact location (URL) in an *http_archive* workspace rule.
- Added property detect.project.inspector.path to enable pointing Synopsys Detect to a local Project Inspector zip file.
- Added property detect.status.json.output.path to place a copy of the status.json file in a specified directory.

Changed features

- Enhancements to error reporting to ensure that any exception will have the root cause reported in the error message for certain exception types.
- Overall Detect exit status is now being reported along with individual detector status/issues in the Status.json file.
- The __MACOSX directory will now be ignored by default when determining which detectors are applicable to a project.

Resolved issues

- (IDETECT-3419) Resolved an issue where the NuGet inspector cannot be found when a solution file cannot be found but multiple C# projects are found by Detect.
- (IDETECT-3306) Resolved an issue where a NullPointerException would occur when project inspector discovered no modules for a project.
- (IDETECT-3307) Warn when project inspector cannot be downloaded, installed, or found.

- (IDETECT-3187) Report Black Duck provided error message (from response body) whenever a Black Duck api call returns an error code
- (IDETECT-3311) Include Detect's "Overall Status" in the status.json / diagnostic zip
- (IDETECT-3449) Resolved an issue that caused overridden violations to be reported as active violations when the BOM contained additional active violations.
- (IDETECT-3476) Resolved an issue that caused an "Input request parsing error" on IaC scans on certain projects when running on Windows.

Version 8.0.0

New features

- Synopsys Detect will now retry (until timeout; see property `detect.timeout`) BDIO2 uploads that fail with a non-fatal exit code.
- Added Detector cascade. Refer to *Detector search and accuracy* for more information.

Changed features

- The default value of `detect.project.clone.categories` now includes DEEP_LICENSE (added to Black Duck in 2022.2.0), raising the minimum version of Black Duck for Synopsys Detect 8.0.0 to 2022.2.0.
- The *codelocation naming scheme* has changed. To prevent old codelocations from contributing stale results to re-scanned projects, set property `detect.project.codelocation.unmap` to true for the first run of Synopsys Detect 8. This will unmap the old codelocations.
- The default value of `detect.force.success.on.skip` has changed to false, so by default Synopsys Detect will exit with return code FAILURE_MINIMUM_INTERVAL_NOT_MET (13) when a scan is skipped because the Black Duck minimum scan interval has not been met.
- By default, all detectors now include in their dependency graph all discovered dependencies, packages, and configurations, because the default for properties `detect.*.[dependency|package|configuration].types.excluded` is NONE. This is a change in the default behavior for the following detector types: GO_MOD, GRADLE, LERNA, RUBYGEMS.
- Dropped NONE as a supported value for the following properties: `detect.included.detector.types`, `detect.tools`.
- Dropped ALL as a supported value for the following properties: `detect.excluded.detector.types`, `detect.tools.excluded`.
- Removed support for parsing SBT report files.
- Cargo project dependency graphs are no longer post-processed to reduce direct dependencies in the BOM.

- Removed the ability to upload BDIO2 documents to legacy endpoints via the `blackduck.legacy.upload.enabled` property.
- Removed the ability to choose the type of BDIO aggregation strategy via the now removed `detect.bom.aggregate.remediation.mode` property. All BDIO will be aggregated in a manner similar to Synopsys Detect 7's SUBPROJECT remediation mode.
- Synopsys Detect now only produces a single Scan in Black Duck for Detectors, named (by default) "<projectName>/<projectVersion> Black Duck I/O Export".
- detect8.sh has improvements (relative to detect7.sh and detect.sh) related to argument handling that simplify its argument quoting/escaping requirements.
- Synopsys Detect requires and runs Detect Docker Inspector version 10.
- Incorporated Detect Docker Inspector documentation into Synopsys Detect documentation.
- The search for files for binary scanning (when property `detect.binary.scan.file.name.patterns` is set) now excludes directories specified by property `detect.excluded.directories`.
- The status.json field `detectors[n].descriptiveName` (which was simply a hyphen-separated concatenation of the `detectorType` and `detectorName` fields) has been removed.
- There is no longer a distinction between extended and non-extended diagnostic zip files. All diagnostic zip files now include all relevant files.
- The following properties (that were deprecated in Synopsys Detect 7.x) have been removed: `blackduck.legacy.upload.enabled`, `detect.bazel.dependency.type`, `detect.bdio2.enabled`, `detect.bom.aggregate.name`, `detect.bom.aggregate.remediation.mode`, `detect.conan.include.build.dependencies`, `detect.detector.buildless`, `detect.docker.path.required`, `detect.dotnet.path`, `detect.go.mod.enable.verification`, `detect.gradle.include.unresolved.configurations`, `detect.gradle.inspector.version`, `detect.lerna.include.private`, `detect.maven.buildless.legacy.mode`, `detect.maven.include.plugins`, `detect.npm.include.dev.dependencies`, `detect.npm.include.peer.dependencies`, `detect.nuget.inspector.version`, `detect.packagist.include.dev.dependencies`, `detect.pear.only.required.deps`, `detect.pnpm.dependency.types`, `detect.pub.deps.exclude.dev`, `detect.ruby.include.dev.dependencies`, `detect.ruby.include.runtime.dependencies`, `detect.sbt.excluded.configurations`, `detect.sbt.included.configurations`, `detect.sbt.report.search.depth`, `detect.yarn.prod.only`.

Resolved issues

- (IDETECT-3375) Resolved an issue where Synopsys Detect would unnecessarily upload empty BDIO entry file when initiating an IaC scan.

- (IDETECT-3224) Resolved an issue where Cargo projects with Cyclical dependencies could cause a failure of Synopsys Detect.
- (IDETECT-3246) Resolved an issue where Synopsys Detect would fail when scanning flutter projects after a new version of flutter was released.
- (IDETECT-3275) Resolved an issue that caused impact analysis to fail with an "Unsupported class file major version" error when an analyzed .class file contained invalid version bytes (byte 7 and 8).
- (IDETECT-3180) Resolved an issue that caused the Binary Search tool to throw an exception when the patterns provided via property detect.binary.scan.file.name.patterns matched one or more directories.
- (IDETECT-3352) Resolved an issue that caused the Gradle Project Inspector detector to fail when the value of detect.output.path was a relative path.
- (IDETECT-3371) Resolved an issue that could cause some transitive dependencies to be omitted from aggregated BDIO in cases where the transitive dependencies provided by the package manager for a component differed across subprojects.

Release notes for supported versions

Version 7.14.0

New features

- Added support for Swift projects built with Swift 5.6 or later.
- Added support for running IaC scans via Synopsys Detect. See *IaC Scan* for more details. Note: IaC capabilities require Black Duck 2022.7.0 or later.

Version 7.13.2

- (IDETECT-3291) Resolved an issue where the NuGet Inspector would only be found for the first applicable detector.
- (IDETECT-3289) Resolved an issue where the NuGet Inspector could not handle Implicit Dependencies in a Package Reference.

Version 7.13.1

- (IDETECT-3286) Resolved an issue that caused the 7.13.0 .jar to be unsigned. The 7.13.1 .jar is signed.

Version 7.13.0

New features

- Added support for a buildless Pipenv detector that parses the Pipfile.lock file (see the *python support page* for more details).
- Synopsys Detect now includes pass-through properties when logging configuration at the beginning of a run.
- Added support for Xcode Workspaces (see the *swift support page* for more details).

Changed features

- Nuget Inspector is now shipped as a self-contained executable and has no runtime requirements.
- Deprecated property detect.detector.buildless, to be replaced with detect.accuracy.required. See *property description* for more details.

Resolved issues

- (IDETECT-3136) Resolved an issue where NPM's `package-lock.json` was prioritized over `npm-shrinkwrap.json`.
- (IDETECT-3184) Resolved an issue that prevented matches for Bazel maven_install components with complex (>3 parts) maven_coordinates values.
- (IDETECT-3207) Resolved an issue that prevented Bazel and Docker Tool issues from being reported in the issues section of the Synopsys Detect log and status file.

Dependency update

- Upgraded to Spring Boot version 2.6.6 / Spring version 5.3.18.

Version 7.12.1

Changed Features

- When signature scanning is skipped due to the minimum scan interval on Black Duck not being met, Synopsys Detect by default will treat the run as a success and will not wait for the skipped

scan(s). If the property detect.force.success.on.skip is set to false, Synopsys Detect will instead return exit code 13 when one or more signature scans were skipped.

Version 7.12.0

New features

- Verified support for Java 16 and 17.
- Added new properties detect.gradle.excluded.project.paths and detect.gradle.included.project.paths to allow filtering on paths which gradle guarantees to be unique.
- Added support for vendoring Go Mod dependencies using *detect.go.mod.dependency.types.excluded=VENDORED* to exclude *test* and *build system* dependencies from Go modules declaring a version prior to `Go 1.16`.
- Added a feature that allows users to configure Synopsys Detect to fail when policies of a certain name are violated. See *detect.policy.check.fail.on.names* for details. Note: this feature requires Black Duck 2022.2.0 or later.
- Added Rapid Compare Mode which enables returning only the differences in policy violations compared to a previous scan.

Changed features

- Changed default value of detect.project.clone.categories from ALL to COMPONENT_DATA, CUSTOM_FIELD_DATA, LICENSE_TERM_FULFILLMENT, VERSION_SETTINGS, VULN_DATA. This avoids the automatic setting of the clone category DEEP_LICENSE introduced in Black Duck 2022.2.0. Users of Synopsys Detect 7.12.0 that wish to pass DEEP_LICENSE or ALL as a value to detect.project.clone.categories must be using Black Duck 2022.2.0 or later.
- Added new property detect.bazel.workspace.rules to replace the now deprecated detect.bazel.dependency.type property.
- For Go Mod projects, successfully executing `go version` is now required. Unsuccessful attempts now result in a run failure.
- The property *detect.go.mod.dependency.types.excluded* now only accepts a single value rather than a list of values.

Resolved issues

- (IDETECT-3016) Resolved an issue where proxies may block HEAD requests made by Synopsys Detect when attempting to download the Signature Scanner from Black Duck. Because the criteria

that Synopsys Detect uses to download the Black Duck Signature Scanner is new, the next run will re-download the Signature Scanner.

- (IDETECT-3165) Resolved an issue that could cause the Bitbake detector to fail with error `Graph Node recipe ... does not correspond to any known layer.`

Version 7.11.1

Changed features

- Updated Synopsys Detect to package Air Gap with the latest Nuget Inspectors: IntegrationNugetInspector:3.1.1, BlackduckNugetInspector:1.1.1, NugetDotnet3Inspector:1.1.1, NugetDotnet5Inspector:1.1.1.

Version 7.11.0

New features

- Added a feature that allows users to set a license for a project version using the property *detect.project.version.license*.
- Added support for identifying dependency relationships between Linux package manager components in images.

Changed features

- The Go Mod Cli Detector no longer uses the "-u" flag when running **go list -m all**. This results in significantly faster scan times against Go Mod projects.
- Deprecated the `detect.pnpm.dependency.types` property in favor of `detect.pnpm.dependency.types.excluded` for property consistency.

Resolved issues

- (IDETECT-2925) Resolved an issue that could cause the Bitbake detector to incorrectly identify the layer of a dependency recipe.
- (IDETECT-3080) Fixed an issue where Synopsys Detect would not include multiple versions of the same package in Cargo projects.

- (IDETECT-3012) Resolved an issue that caused Synopsys Detect to incorrectly use BLACKDUCK_USERNAME and BLACKDUCK_PASSWORD.

Version 7.10.0

New features

- Added support for the Apache Ivy package manager.
- Build dependencies can now be excluded from BitBake results.

Changed features

- Synopsys Detect now classifies empty code location warning messages as the DEBUG logging level instead of the previous classification as the WARN logging level.
- BitBake detector: Added support for BitBake 1.52 (Yocto 3.4).
- BitBake detector: Added support for BitBake projects with build directories that reside outside the project directory.
- Deprecated many properties relating to filtering dependency types from the BOM. These property replacements will reduce the number of properties, apply consistency to detector properties, add filtering abilities, and overall simplify Detect configuration.
  - Deprecated the following properties:
    - detect.conan.include.build.dependencies
    - detect.pub.deps.exclude.dev
    - detect.go.mod.enable.verification
    - detect.gradle.include.unresolved.configurations
    - detect.lerna.include.private
    - detect.npm.include.dev.dependencies
    - detect.npm.include.peer.dependencies
    - detect.packagist.include.dev.dependencies
    - detect.pear.only.required.deps
    - detect.ruby.include.runtime.dependencies
    - detect.ruby.include.dev.dependencies
    - detect.yarn.prod.only
  - Added the following replacement properties:

- detect.conan.dependency.types.excluded
- detect.pub.dependency.types.excluded
- detect.go.mod.dependency.types.excluded
- detect.gradle.configuration.types.excluded
- detect.lerna.package.types.excluded
- detect.npm.dependency.types.excluded
- detect.packagist.dependency.types.excluded
- detect.pear.dependency.types.excluded
- detect.ruby.dependency.types.excluded
- detect.yarn.dependency.types.excluded

Resolved issues

- (IDETECT-2949) Fixed an issue where Synopsys Detect failed to properly parse Go module version names containing '-' characters.
- (IDETECT-2959) Fixed an issue where Synopsys Detect would not fail when running `go mod why` fails.
- (IDETECT-2971) Fixed an issue where Synopsys Detect would not produce unique code location paths for Pnpm projects.
- (IDETECT-2939) Fixed an issue where NPM projects that had no declared dependencies would not exclude peer or dev dependencies.
- (IDETECT-3038) Fixed an issue where Synopsys Detect would fail to parse file dependency declarations for pnpm projects in their pnpm-lock.yaml files.
- (IDETECT-3000) Fixed an issue where Synopsys Detect would error out when a user's source directory and output directory did not share a common root.

Version 7.9.0

New features

- Added support for the Xcode Swift Package Manager for Xcode projects using the built-in *Swift Packages* feature.
- Added detect.bdio.file.name to specify the name of the output BDIO file.
- Added system architecture DEBUG level logs to assist with support.

Changed features

- The version of each package manager tool executed by CLI detectors is now logged at DEBUG level.

Resolved issues

- (IDETECT-2499) Fixed an issue in the Gradle Inspector that caused it to exclude all identically named subprojects except one.
- (IDETECT-2953) Fixed the project and project version links in risk report.
- (IDETECT-2989) Fixed an issue with Go Mod projects where Synopsys Detect included unused transitive dependencies, despite detect.go.mod.enable.verification being set to 'true'.
- (IDETECT-2935) Verified that Synopsys Detect is compatible with Gradle version 7.X.

Version 7.8.0

New features

- Added support for the pnpm package manager.
- Added property detect.project.group.name for setting the Project Group.
- Synopsys Detect now falls back to using a previously downloaded Docker Inspector, Project Inspector, and/or NuGet Inspector when https://sig-repo.synopsys.com is unreachable.

Version 7.7.0

New features

- Added support for uploading rapid scan config file when a file named '.bd-rapid-scan.yaml' is present in the source directory.
- Added the property detect.project.inspector.arguments for providing additional arguments to the project inspector across all invocations.

Resolved issues

- (IDETECT-2808, IDETECT-2863) Resolved an issue where Synopsys Detect would incorrectly resolve relative paths when processing signature scan targets.
- (IDETECT-2859) Resolved an issue where Synopsys Detect was using an outdated cookie spec when making a request, resulting in a warning message.

Version 7.6.0

New features

- Added the property *detect.follow.symbolic.links* which can be used to enable Synopsys Detect to follow symbolic links when searching directories for detectors, when creating exclusions for signature scan, and when creating binary scan targets.
- Added support for Open Container Initiative (OCI) images provided to Synopsys Detect using the *detect.docker.tar* property.
- Added the property *detect.gradle.include.unresolved.configurations* for toggling the inclusion of *unresolved Gradle configurations*.
- Added Project Inspector support for MAVEN and GRADLE when resolving buildless dependencies.
- Added support for NUGET buildless using the Project Inspector.

Changed features

- The detect.project.clone.categories property now supports ALL and NONE as options.
- The the default value for property detect.project.clone.categories has changed to ALL.
- Deprecated detect.bom.aggregate.name, detect.bom.aggregate.remediation.mode, and blackduck.legacy.upload.enabled. In version 8, Synopsys Detect will only operate in SUBPROJECT aggregation mode to report the dependency graph with greater accuracy.
- Maven defaults to the legacy buildless parser, Project Inspector must be enabled with detect.maven.buildless.legacy.mode. In version 8, it will default to Project Inspector.
- Deprecated detect.maven.include.plugins as Project Inspector does not support plugins. In version 8, we will only support the Project Inspector Maven implementation which will have its own configuration mechanism.
- The Air Gap Zip generation options no longer support individual package managers. Instead, either a FULL air gap can be created, or a NO_DOCKER air gap can be created. This is to help support project inspector which spans multiple package managers.

Resolved issues

- (IDETECT-2834) Resolved an issue where GoMod components missing a version were not being properly filtered causing a NullPointerException.
- (IDETECT-2829) Resolved an issue that caused Synopsys Detect to use the wrong scan cli when in offline mode and ignore a specified local scan cli.
- (IDETECT-2820) Resolved an issue where pypi components in conda projects were not being matched.
- (IDETECT-2773) Resolved an issue where Synopsys Detect was not replacing module paths as specified in go mod replace statements.

Version 7.5.0

New features

- Added support for the Dart package manager.

Changed features

- The following directories are no longer excluded from Signature Scan by default: bin, build, out, packages, target. .synopsys directories are now excluded from both Detector search and Signature Scan.
- The Docker Inspector can now be included (using the detect.tools property) when using the *rapid scan mode*.
- Instead of "lite" Docker images that automatically disable all detectors, Synopsys Detect now supports "buildless" Docker images that automatically disable detectors that depend on the presence of build tools but leave buildless detectors enabled.

Resolved issues

- (IDETECT-2830) Resolved an issue that caused the Gradle detector to fail when run in air gap mode.
- (IDETECT-2816) Resolved an issue that caused a "Duplicate key" error when running binary scan on multiple files with the same name.

Version 7.4.0

New features

- Added SUBPROJECT remediation mode, invoked using property detect.bom.aggregate.remediation.mode. Use only with Black Duck 2021.8.0 or later.

Version 7.3.0

New features

- Added support for the Carthage package manager.

Changed features

- The Poetry detector is no longer categorized as a PIP detector and is now categorized under detector type POETRY.
- Simplified the property deprecation lifecycle to the following: Use of deprecated properties will result in logged warnings until the next major version release, at which time those properties will be removed from Detect (and ignored if used). Properties that were deprecated in Detect 6.x have been removed in this release. Properties deprecated in Detect 7.x will be removed in Detect 8.0.0.

Version 7.2.0

Changed features

- Improved the readability of Rapid mode results.

Resolved issues

- (IDETECT-2532) Resolved an issue that could cause multiple versions of Go-Mod dependencies to appear in the BOM.
- (IDETECT-2668) Resolved an issue that caused Go-Mod dependencies with a replacement version to be omitted.

- (IDETECT-2722) Resolved an issue that caused the version to be omitted from Go-Vendr dependencies when the vendor.conf separated dependency name and version with multiple space characters.
- (IDETECT-2672) Resolved an issue that could cause the Black Duck access token to appear in the log.
- (IDETECT-2739) Resolved an issue that caused the default to be used when the provided Risk Report path did not exist.

Version 7.1.0

New features

- Added ability to specify custom fonts to be used during risk report generation. See *here* for more details.
- There now exist Docker images that can be used to run Synopsys Detect from within a container. See *Running Synopsys Detect from within a Docker container* for more details.
- Added detect.go.mod.enable.verification for disabling the `go mod why` check that Synopsys Detect uses to filter out unused dependencies.
- Added support for dotnet 5 when running the NuGet inspector.
- Added a new property *detect.npm.include.peer.dependencies* which allows the users to filter out NPM peer dependencies from their BOM.

Changed features

- The following clone categories were added to the default value for property detect.project.clone.categories: LICENSE_TERM_FULFILLMENT, CUSTOM_FIELD_DATA
- The "Git Cli" detector has been renamed to the "Git" detector.
- Whenever Synopsys Detect runs the following tools, it now logs (at level DEBUG) the tool's version: git, gradle, maven, conan, pip, and python.

Resolved issues

- (IDETECT-2541) Resolved an issue that caused the CLANG detector to fail when run in non-English locales on Ubuntu and Debian systems.
- (IDETECT-2505) Resolved an issue that caused go mod components with +incompatible version suffixes to not be matched on Black Duck.

- (IDETECT-2629) Resolved an issue that caused go mod projects without source having an empty BOM with the introduction of the detect.go.mod.enable.verification property.
- (IDETECT-2659) Resolved an issue that caused Synopsys Detect to falsely report a missing detector when that detector matched only at a depths > 0 and was included in the value of property detect.required.detector.types.
- (IDETECT-2696) Resolved an issue that could cause Synopsys Detect to fail with "IllegalStateException: Duplicate key {codelocation name}" when creating >100 codelocations in one run.
- (IDETECT-2659) Resolved an issue that caused Detect to falsely report "One or more required detector types were not found" when the required detector ran based on files found in a subdirectory.
- (IDETECT-2541) Resolved an issue that caused the CLANG detector to fail with "Unable to execute any supported package manager" when run with a non-English locale on an alpine system.

Version 7.0.0

New features

- Added scripts detect7.sh and detect7.ps1 for invoking Synopsys Detect 7.x.x. detect.sh and detect.ps1 will (by default) continue to invoke the latest Synopsys Detect 6 version.
- Added support for Yarn workspaces.
- Added support for the dependency graph SBT plugin. Resolution cache generation is no longer a requirement of the SBT detector.
- Added the properties *detect.excluded.directories*, *detect.excluded.directories.defaults.disabled*, and *detect.excluded.directories.search.depth* to handle exclusions for detector search and signature scanning.
- Added ability to specify excluded directory paths using *glob patterns*.
- Added properties *detect.lerna.excluded.packages* and *detect.lerna.included.packages* to exclude and include specific Lerna packages.
- Added critical security risks to the Black Duck Risk Report pdf.
- Added detect.target.type to enhance the docker user experience. When set to IMAGE, some tools are automatically disabled and detect optimizes for an image-based scan.
- Added binary scanning of the container filesystem to the default Docker image scanning workflow. If you are scanning Docker images and your Black Duck server does not have the binary scanning feature enabled; use --detect.tools.exluded=BINARY_SCAN to disable the binary scan step.

Changed features

- The following directories will be excluded from signature scan by default, in addition to node_modules: bin, build, .git, .gradle, out, packages, target. Use *detect.excluded.directories.defaults* to disable these defaults.
- Detect no longer supports the exclusion of individual files during detector search, only directories.
- Gradle detector no longer uses the gradle inspector. Only the init script is required.
- The default BDIO format for communicating dependency graphs to Black Duck has been changed from BDIO1 to BDIO2.
- Risk report generation will download fonts from Artifactory or use the font files in the fonts directory in the air gap zip of detect.

Resolved issues

- (IDETECT-2462) Resolved an issue where projects were being inaccurately diagnosed as Poetry projects due to the presence of a pyproject.toml file.
- (IDETECT-2527) Resolved an issue in the Go Mod detector to extract and process data even if 'go mod why' command fails to run.
- (IDETECT-2434) Resolved an issue in the CLANG detector on Ubuntu and Debian systems that caused it to omit a package when that package had been installed on the system from multiple architectures.
- (IDETECT-2362) The CLANG detector now uses the KB preferred alias namespace feature for improved match accuracy.
- (IDETECT-2413) Resolved an issue to upgrade internal dependencies to support JDK 15.
- (IDETECT-2409) Resolved an issue to allow Gradle detector to support Gradle 6.8.
- (IDETECT-2099) Improved error reporting for exceptions that occur during a Detect run. For each exception, a Detect "issue" is written to the log and to the status.json file.
- (IDETECT-2516) Improved error reporting for the case where environment variable BDS_JAVA_HOME is set incorrectly.

Release notes for older versions

Version 6.9.1

Resolved issues

- (IDETECT-2555) Resolved an issue that could cause Detect, when run against Black Duck 2020.10.0, to fail with a message like: "Cannot cast... to... VersionBomCodeLocationBomComputedNotificationUserView".

Version 6.9.0

New features

- Added ability for detectors to explain why they applied. It will appear in the logs at info level and in the status.json.
- Added the property detect.binary.scan.search.depth to define the directory search depth for the binary scanner.
- The status.json file now features a list of the provided Detect property values.
- When Detect is not configured to connect to Black Duck or run offline, a link to the Detect help is included in an error message.

Changed features

- Added the timezone to the date format in the default log message format.
- Reverted deprecations for detect.blackduck.signature.scanner.arguments, detect.blackduck.signature.scanner.copyright.search, detect.blackduck.signature.scanner.dry.run, detect.blackduck.signature.scanner.individual.file.matching, detect.blackduck.signature.scanner.license.search, detect.blackduck.signature.scanner.local.path, detect.blackduck.signature.scanner.paths, detect.blackduck.signature.scanner.snippet.matching, detect.blackduck.signature.scanner.upload.source.mode.

Resolved issues

- (IDETECT-1986) Resolved an issue where warnings regarding reflective access appear at the start of Detect.
- (IDETECT-2400) Resolved an issue where 'dependencies' would be removed from the value of the detect.gradle.build.command property.
- (IDETECT-2394) Resolved an issue that created inaccurate relationships in the BDIO files when Gemlock files were processed.

- (IDETECT-2404) Resolved an issue where signature scanner arguments passed through detect.blackduck.signature.scanner.arguments that contained space were being improperly parsed.
- (IDETECT-2525) Resolved an issue with the Yarn detector that caused component version information to be missing when the yarn.lock file contained quoted field keys.
- (IDETECT-2254) Resolved an issue with the Yarn detector that caused certain components to be omitted from some Yarn 2 projects.
- (IDETECT-2471) Resolved an issue where a missing Git executable in certain situations causes an exception.

Version 6.8.0

New features

- Added support for Conan projects that have the Conan revisions feature enabled.
- Added detect.pip.path for advanced users who wish to specify which pip executable to run.
- Improved the Pip Inspector to attempt to discover files named "requirements.txt" if no requirements files are specified through detect.pip.requirements.path.

Changed features

- Added detect.timeout to consolidate the functionality of blackduck.timeout and detect.report.timeout.
- Added date of latest scan for a project version to the risk report pdf.
- Deprecated properties blackduck.timeout and detect.report.timeout. They have been consolidated into the new property detect.timeout.
- Deprecated all Detect exclusion properties. Future releases will feature a new property to extend and consolidate these properties.
- Deprecated all Detect signature scanner properties. Future releases will feature an alternative mechanism for providing signature scanner arguments to Detect.
- Deprecated property detect.resolve.tilde.in.paths. Resolving tildes is a shell feature which Detect will no longer support in a future version.
- Deprecated property detect.python.python3. Due to the January 2020 sunset of Python 2, this property (which toggles between searching for a 'python' and 'python3' executable) is no longer necessary. See: *PEP-394*
- Deprecated properties detect.docker.inspector.air.gap.path, detect.gradle.inspector.air.gap.path, and detect.docker.inspector.air.gap.path as part of an effort to simplify Detect.

- Deprecated properties detect.default.project.version.scheme, detect.default.project.version.text, detect.default.project.version.timeformat as part of the effort to simplify Detect.
- Deprecated properties blackduck.username and blackduck.password. Authentication should be performed using an API token.

Resolved issues

- (IDETECT-2216) Resolved an issue that prevented non-ASCII filenames from being correctly transmitted to Black Duck during a binary scan file upload.
- (IDETECT-2227) Resolved an issue where Nuget Inspectors would parse source files for assembly version.
- (IDETECT-2281) Resolved an issue that included golang dependencies that were not linked in the compiled go application. *241*
- (IDETECT-2294) Resolved an issue where Git credentials could be logged when reading the remote URL.
- (IDETECT-2296) Resolved an issue wherein the Pip Inspector would cease parsing a requirements file if it encountered a dependency which it could not resolve.
- (IDETECT-2276) Resolved an issue that caused the CLANG detector to omit components for which multiple architectures are installed.

Version 6.7.0

Resolved issues

- (IDETECT-2285) Resolved an issue that could cause Detect to fail to authenticate with Black Duck with the error message "No Bearer token found when authenticating.".
- (IDETECT-2221) Resolved an issue where the Docker Inspector logging level was not set correctly when property logging.level.detect was used.
- (IDETECT-2213) Resolved an issue that could cause the CLANG detector to omit some components on Debian-based Linux systems.
- (IDETECT-2284) Resolved an issue that could cause the CLANG detector to omit some components for projects using the clang/clang++ compiler when source files reference include files using non-canonical paths.
- (IDETECT-2216) Resolved an issue that caused non-ASCII characters in binary scan metadata (filename, code location name, project name, and version name) to be converted to '?' characters when submitted to Black Duck.
- (IDETECT-2291) Reverted replacement data support. Detect will report exactly what Gradle reports. This reverts IDETECT-2038, IDETECT-2203.

- (IDETECT-2241) Resolved an issue where platform dependent cocoapods will throw an exception when they are not installed.
- (IDETECT-2289) Resolved an issue that could cause Black Duck API token-based authorization to fail with "411 Length Required" HTTP status when communicating with Black Duck through a proxy.

Version 6.6.0

Changed features

- The Docker Inspector now works on Windows 10 Enterprise.
- Upon connecting to Black Duck, the users' roles and groups, which are only used in DEBUG-level logging, are no longer fetched unless logging level is DEBUG or higher.
- The error messages produced for binary scan file upload failures have been improved.
- The "detectors" field in the status.json file now features status data with more-expressive error codes derived from the runtime class of a detectable result.
- Detect will follow 308 redirects when communicating with Black Duck.

Resolved issues

- (IDETECT-2038, IDETECT-2203) Resolved an issue where the Gradle Inspector would produce false positives in Gradle because of dependency replacement from the root project.
- (IDETECT-2180) Resolved and issue where the Pip Inspector would fail against requirements.txt files generated by the pip-compile tool.
- (IDETECT-2108) Resolved an issue where Lerna packages where being reported as missing dependencies.
- (IDETECT-2138, IDETECT-2161, IDETECT-2172) Resolved issue where Gradle parse detector would fail due to an inability to resolve classes, referenced in the project's build scripts, that were outside of Detect's classpath.
- (IDETECT-2110) Nuget inspectors will correctly return -1 when an error occurs by default.
- (IDETECT-2202) Impact analysis code locations will now appear in the status.json file.

Known issues

- When running the Docker Inspector on Windows, Synopsys Detect may fail to clean up all its working directories (and log the message "Error trying cleanup") due to the following Docker issue: https://github.com/docker/for-win/issues/394.

- False positives from Gradle are still possible if the replacement dependency is defined within a subproject that has subprojects. Work is being continued to fix this with IDETECT-2218.

Version 6.5.0

New features

- Added *properties* for enabling diagnostic mode.
- Synopsys Detect now supports Vulnerability Impact Analysis. Enabled using *Vulnerability Impact Analysis Enabled* property.

Changed features

- Enabling diagnostic mode is now controlled through two new properties.
- *--detect.diagnostic*
- *--detect.diagnostic.extended*
- The *detect.bazel.dependency.type* property now accepts a comma-separated list of dependency types, or the value *NONE*, or the value *ALL*.

Resolved issues

- (IDETECT-2054) Resolved an issue that caused the Gradle Inspector to fail when detect.output.path is set to a relative path.

Version 6.4.2

Resolved issues

- (IDETECT-2164) Resolved an issue with scanning Go applications when using the go list -m command, which couldn't determine available upgrades using the vendor directory.

Version 6.4.0

New features

- Bazel detector: added support for Bazel projects that specify dependencies using the haskell_cabal_library repository rule.
- NuGet detector: added support for DotNet 3.1 runtime.
- Synopsys Detect now supports projects managed by the Lerna package manager.
- Synopsys Detect now supports projects managed by the Cargo package manager.
- Synopsys Detect now supports projects managed by the Poetry package manager.

Changed features

- Eliminated any need for the Black Duck Global Code Scanner overall role.
- The CLANG detector collects any dependency files not recognized by the Linux package manager that reside outside the source directory (the directory containing the compile_commands.json file) and writes them to the status.json file.
- Added the property *detect.blackduck.signature.scanner.copyright.search*.
- Removed PipEnv from the list of buildless detectors as it was never buildless.
- Improved output for signature scanner status and included descriptions for exit codes when reporting overall status.
- Status.json file now collects code location data generated by all tools, not just detectors.
- Status.json file now collects issue data generated by all tools.

Resolved issues

- (IDETECT-2019) Resolved an issue where the pip inspector would not be able to parse the requirements.txt file if pip's version was >= 20.1.
- (IDETECT-2034) Resolved an issue that would cause a NullPointerException when Synopsys Detect's initial attempt at generating a code location name produced a code location name greater than 250 characters and either code location prefix or code location suffix is not set.
- (IDETECT-1979) Resolved an issue that could cause the CLANG detector to miss some dependencies because it failed to correctly parse complex nested quoted strings within compile_commands.json values.
- (IDETECT-1966) Resolved an issue that would cause Detect to ignore replacement directives for Go Mod projects.

Known Issues

- When a Lerna package depends on another Lerna package within the project, an error may appear indicating a missing dependency on that package. This is normal and no dependencies are missing. This will be fixed in a future release.

Version 6.3.0

New features

- The Yarn detector now extracts project information from package.json files. Git is no longer the default supplier of project information for Yarn projects.
- Added Yarn Detector support for dependencies that are missing a fuzzy version in a lockfile dependency declaration.
- Synopsys Detect logs policy violations when it is configured to *fail on policy violations*.

Changed features

- Users can *upload source* files when *license search* is enabled regardless of whether *snippet matching* has been enabled.
- Synopsys Detect is now compatible with Yocto 3.0.
- Synopsys Detect stops if the Docker Inspector tool applies and Synopsys Detect is running on Windows.
- Synopsys Detect configures Docker Inspector's working directories inside Synopsys Detect's run directory.
- Synopsys Detect requires and runs Docker Inspector version 9.
- Moved the location to which detect.sh downloads the Synopsys Detect .jar from /tmp to ~/synopsys-detect/download.

Resolved issues

- (IDETECT-1906) Resolved an issue wherein git extraction might fail if "git log" returned unexpected output. As a last resort, the commit hash will be used as a version.
- (IDETECT-1883) Resolved an issue where Synopsys Detect failed to extract project information when parsing a Git repository with a detached head while in buildless mode.

- (IDETECT-1970) Resolved an issue where the default value for *parallel processors* was not used. The available runtime processor count was being used instead.
- (IDETECT-1973) Resolved an issue where the NuGet exe inspector would not resolve from Artifactory.
- (IDETECT-1965) Resolved an issue where Synopsys Detect would fail to resolve environment variables where it did so previously.
- (IDETECT-1974) Resolved an issue wherein the Yarn detector was throwing an exception for dependencies not defined in the yarn.lock file.
- (IDETECT-2037) Resolved an issue where Synopsys Detect would fail with a "hostname in certificate didn't match" error while downloading the Gradle inspector.

Version 6.2.1

Resolved issues

- Resolved an issue wherein an exception was thrown when generating a risk report if users didn't set the risk report output path explicitly. (IDETECT-1960)

Version 6.2.0

New features

- The Synopsys Detect .jar file is now signed, enabling *code verification* by users.
- *Simple proxy information* will be forwarded to the Gradle Inspector.
- Detect now creates a status file describing the results of the run which includes things like *issues, results, and status codes.*
- The property configuration table in the log now includes the origin of the property's value.
- Added the property *detect.blackduck.signature.scanner.license.search*.
- Added the property *detect.blackduck.signature.scanner.individual.file.matching*.
- If an executable returns a nonzero exit code, Detect will now log the executable output automatically.
- Added page for deprecated properties in help.
- Detect-generated risk reports now feature Synopsys logo and branding.

Changed features

- The PipEnv Detector now parses a json representation of the dependency tree.
- Powershell download speed increased.

Resolved issues

- Resolved an issue where the download URL for Synopsys Detect was being set to an internal URL upon release (IDETECT-1847).
- Resolved an issue where all transitive dependencies found by the Pip inspector were being reported as direct dependencies (IDETECT-1893).
- Resolved an issue where using pip version 20+ with the Pip inspector caused a failure to import a dependency. *GitHub PR* (IDETECT-1868)
- Resolved the following vulnerabilities (IDETECT-1872):
- org.springframework.boot:spring-boot-starter 5.1.7.RELEASE BDSA-2020-0069 (CVE-2020-5398)
- Resolved an issue where Synopsys Detect had the potential to fail on projects that utilized Yarn workspaces (IDETECT-1916).
- Note: Yarn workspaces are not currently supported. See *yarn workspace support*.
- Resolved an issue in the Bazel Detector that caused it to fail for the maven_install rule when the tags field contained multiple tags with a mixture of formats (IDETECT-1925).
- When parsing package.xml files, Detect will no longer raise a SAXParseException when the file contains a doctype declaration, and will continue parsing the rest of the file (IDETECT-1866).
- Resolved an issue that could cause generation of an invalid Black Duck Input/Output (BDIO) file when the only differences between two component names/versions are non-alphanumeric characters (IDETECT-1856).

Version 6.1.0

New features

- Added the property *detect.bdio2.enabled*.
- Added the property *detect.pip.only.project.tree*.
- Added the property *detect.bitbake.search.depth*.
- Added the property *detect.bazel.cquery.options*.
- Added the property *detect.docker.image.id*.

- Added the property *detect.docker.platform.top.layer.id*.
- Added the property *detect.bom.aggregate.remediation.mode*

Changed features

- Deprecated all Polaris properties.
- Added wildcard support for several include/exclude list properties.
- Improved the structure of the dependency information produced by the Yarn detector by changing its approach. It now parses dependency information from yarn.lock and package.json, instead of running the yarn command. Since the yarn command is no longer executed, the detect.yarn.path property has been removed.
- Improved match accuracy for Bitbake projects by improving external ID generation for dependencies referenced using Git protocols, and dependencies referenced with an epoch and/or revision.
- Improved the reliability of the Bitbake detector by generating recipe-depends.dot and package-depends.dot files the source directory, instead of a temporary directory.
- Changed the logging level of Polaris CLI output from DEBUG to INFO.
- Added support for the Noto-CJK font (for Chinese, Japanese, and Korean text) in the risk report.

Resolved issues

- Resolved an issue that can cause a Null Pointer Exception on Maven projects configured for multi-threaded builds.
- Resolved an issue that can cause Detect to fail due to an expired Black Duck bearer token.
- Resolved an issue that causes Detect to fail when a parent project and version are specified, and the project is already a child of the specified parent.
- Resolved an issue that causes Detect to log the git username and password when a git command executed by Detect fails.
- Resolved an issue that can cause Detect to generate a new code location (scan) when the character case of the value of the detect.source.path property differs from a previous run on the same project.
- Resolved the following vulnerabilities: commons-beanutils:commons-beanutils 1.9.3 / BDSA-2014-0129 (CVE-2019-10086), org.apache.commons:commons-compress 1.18 / BDSA-2019-2725 (CVE-2019-12402)

Version 6.0.0

New features

- Added the property detect.binary.scan.file.name.patterns.
- Added the property detect.detector.search.exclusion.files which accepts a comma-separated list of file names to exclude from the Detector search.
- Custom arguments for the source command can now be supplied to Detect through the property detect.bitbake.source.arguments which accepts a comma-separated list of arguments. (1614)
- Added support for the Swift package manager.
- Added support for GoGradle.
- Added support for Go Modules.
- The property detect.pip.requirements.path is now a comma-separated list of paths to requirements.txt files. This enables you to specify multiple requirements files. Each requirements file displays as a new code location in Black Duck.
- Detect now logs username, roles, and groups for the current user.
- Detect now includes the project name/version in every code location name.
- Detect now takes in a go path but does not take in go.dep.path; nor does Detect trigger on *.go.
- The property detect.parallel.processors is added. This property controls the number of parallel threads, and replaces the properties detect.blackduck.signature.scanner.parallel.processors and detect.hub.signature.scanner.parallel.processors.
- Added the property detect.maven.included.scopes. This is a comma-separated list of Maven scopes. Output is limited to dependencies within these scopes, and is overridden by exclude.
- Added the property detect.maven.excluded.scopes. This is a comma-separated list of Maven scopes. Output is limited to dependencies outside these scopes, and is overridden by include.
- Bazel detector: added support for dependencies specified using the maven_install workspace rule. The detect.bazel.advanced.rules.path property is removed.
- When using Detect for static analysis, you can pass the build command to let the Polaris CLI know how to analyze a given project.

Changed features

- Architecture is no longer included in BitBake dependencies discovered by Detect. The property detect.bitbake.reference.impl is no longer used and is deprecated.
- The BitBake detector no longer uses the property detect.bitbake.reference.impl because architecture is no longer required to match with artifacts in the KnowledgeBase. The Bitbake detector now attempts to determine the layer in which a component originated instead of the architecture.
- Improved the Detect on-screen logging to be more concise.

- The PiP inspector is no longer deprecated and is currently supported.
- When creating an air gap zip of Detect using the switch -z or --zip, the created zip file is now published to your output directory.
- Scripts no longer fail if the Artifactory server is unavailable.
- Enhanced placement and formatting of deprecation logs.
- Added support for Java version 11.
- The following properties are removed in Detect version 6.0.0:
- detect.go.dep.path
- detect.npm.node.path
- detect.perl.path
- detect.go.run.dep.init
- detect.maven.scope
- detect.bazel.advanced.rules.path

Resolved issues

- Resolved an issue wherein the Windows Java path construction did not account for direction of the slash. The shell script now uses the correct slash direction, based on the operating system on which Detect is running.
- Resolved an issue wherein Detect was not finding the file recipe-depends.dot written to the current directory. Detect now looks in the source directory to a depth of 1 if it cannot find the expected files in the expected location.
- Resolved an issue wherein Detect was failing if it could not resolve placeholders.
- Resolved an issue wherein Detect was not handling SSH URLs, which caused Detect to fail in extracting project information from the Git executable. GitCliDetectable now properly handles SSH URLs.
- Resolved an issue wherein the Detect JAR was downloading for each scan when the script could not communicate with Artifactory. Now, if the script cannot communicate with Artifactory, and there is an existing downloaded Detect, then the previously downloaded version of Detect runs. However, if you provided a DETECT_LATEST_RELEASE_VERSION and Detect cannot communicate with Artifactory, Detect will not run.
- Resolved an issue wherein Detect was not properly parsing GIT URLs such as git:// git.yoctoproject.org/poky.git.

Version 5.6.2

Resolved issues

- Synopsys Detect version 5.6.2 is a rebuild of version 5.6.0 and 5.6.1 to address an issue with the binary repository to which it was published.

Version 5.6.0

New features

- You can now set custom fields on created Black Duck projects.
- Detect can now generate its own air gap zip.
- Detectors now nest by default.
- Added support for Gradle Kotlin.
- Added support for wildcard (*) in the Detect flag blackduck.proxy.ignored.hosts.
- Added support for --detect.project.tags.
- Added the properties --detect.parent.project.name and --detect.parent.project.version.name.
- Added the property --detect.clone.project.version.latest=true which takes precedence over the exact version name.
- Added support for Yocto 2.0.0.
- Added support to parse components from the <plugins> block in pom.xml. This only works when detect.detector.buildless=true.
- Added capability to represent '' and "" as a null value in Detect multiselect custom fields.

Changed features

- You can now specify the search depth for buildless mode.
- Updated the help menu and provided more detailed help options.
- Diagnostics now includes signature scanner log files.
- Re-enabled empty aggregate file generation.
- Polaris no longer runs the -w switch enabled by default. To retrieve the issue/policy count, you can use the -w switch.
- Match accuracy for Docker images is improved by running the signature scanner on a squashed version of the Docker image instead of the container file system. This results in a different name for the code location because the name of the file being scanned is different. For existing projects, the old code location named by default as <repo>_ <tag>_containerfilesystem.tar.gz/<repo>/<tag>

scan must be removed to ensure it does not contribute stale data to the BOM. Due to the new method of scanning, the code location name has changed. You must remove the old code location in favor of the new code location.

Resolved issues

- Resolved an issue that could cause code location names to contain relative file paths when the value of detect.source.path uses symbolic links to specify the source directory.
- Resolved an issue that caused detect.sh to fail when Java is not on the system path, and the JAVA_HOME path contains a space.
- Resolved an issue wherein the signature scanner may not have been reporting failures correctly.
- Resolved an issue wherein Detect was not locating the file recipe-depends.dot when it was written to the current directory. Detect now searches for the recipe-depends.dot file to a depth of 1 when extracting on a BitBake project.
- Detect no longer fails if the Git executable is not found.
- Resolved an issue wherein Detect may fail when the directory pointed to by --detect.notices.report.path does not exist.

Version 5.5.1

Resolved issues

- Resolved an issue wherein the Pipenv detector was omitting project dependencies.

Version 5.5.0

New features

- Added support for snippet modes.
- The property detect.wait.for.results has been added to wait for Black Duck. The default value is false. If this property is set to true, Detect won't complete until the normal timeout is reached or the underlying systems with which Detect is communicating are once again idle and ready to receive more data. The timeout value is controlled by blackduck.timeout.
- The shell script and PowerShell script now accept DETECT_JAVA_PATH and DETECT_JAVA_HOME as environment variables for pointing to your Java installation.

- Added a new property --detect.detector.search.exclusion.paths. A comma-separated list of directory paths to exclude from a detector search. For example, foo/bar/biz only excludes the biz directory if the parent directory structure is 'foo/bar/'.
- Detect now uses Git information to determine the default project and version names.
- There is a new Detect property for overriding the Git executable: detect.git.path.

Resolved issues

- Resolved an issue that caused the risk report to be generated with invalid links to Black Duck components.
- Resolved an issue that caused a null pointer exception error when a golang's Gopkg.lock file contained zero projects.
- Resolved an issue wherein the Clang detector could omit the epoch from the version string in RPM packages.
- Resolved an issue wherein with two users running Detect on a single system may result in a Permission denied error.
- Resolved an issue wherein the property -detect.policy.check.fail.on.severities may not be waiting for the snippet scans to complete.
- Resolved an issue wherein the property --detect.blackduck.signature.scanner.exclusion.name.patterns may not be following the paths.
- Resolved an issue wherein Detect may fail when the directory specified by --detect.risk.report.pdf.path did not exist. Detect now attempts to create the directory structure to the specified path. A warning is logged if Detect fails to create the directory.
- Resolved an issue wherein properties that had a primary group and additional property group may have been excluded from the group search.
- Resolved an issue wherein the deprecation warning displayed when the deprecated property was provided by the user.
- Resolved an issue with aggregate BOM filename generation that could cause the message Unable to relativize path, full source path will be used to display in the log.
- Resolved an issue that could cause components to be omitted from the BOM for Conda projects.
- Resolved an issue that could cause errors during parsing of Maven projects with long subproject names.

Changed features

- The default value for the property detect.docker.path.required is now false.
- The ALL logging level is replaced with the TRACE logging level.
- The results URL for the Black Duck project BOM is now moved to the Detect Results panel.

- Renamed Detect Results to Detect Status.
- Previously, a temp file remained which could contain plain text user name or password information. This temp file is now removed.
- Bazel is added as an acceptable value to the detect.tools properties.
- Detect now uses the current version of Docker Inspector. This means that no matter what version of Docker Inspector is currently released, Detect now uses that version.

Version 5.4.0

New features

- Added buildless mode.
- Added a new property for BitBake to remove Yocto reference implementation characters.
- Added a new property for adding group names to projects.
- Added a new property for uploading source files.
- Added the additional_components placeholder.

Resolved issues

- Resolved an issue wherein Yarn may have been incorrectly calculating the tree level.
- Resolved an issue wherein Detect may fail when Polaris is excluded, a Polaris URL is provided, and connection to Polaris failed.
- Resolved an issue that caused Detect to follow symbolic links while searching directories for files.
- Resolved an issue wherein Detect was not failing policy for UNSPECIFIED when fail on severities is set to ALL.
- Resolved an issue that could cause a counter (an integer intended to ensure uniqueness), to be unnecessarily appended to a code location name.
- Resolved an issue that may have caused the package manager name to be excluded from the code location name when a code location name was provided.
- Resolved an issue that could cause Detect to continue after a Polaris connection failure.
- Resolved an issue wherein the Detect scan results may incorrectly show development dependencies.
- Resolved an issue that could cause reports to fail due to timeout intermittently.
- Resolved an issue that could cause the value of --polaris.access.token to be logged to the console when detect.sh is invoked.

- Resolved an issue wherein Detect was cleaning up the contents but not the directory of the run.

Changed features

- For getting all logs, the ALL logging level is now TRACE.
- Improved the error message logged when the property detect.binary.scan.file.path, which must point to a readable file, points to something other than a readable file, such as a directory.
- Changed the environment variable used to tell the Detect scripts where to download the Detect jar. The previous value DETECT_JAR_PATH is now changed to DETECT_JAR_DOWNLOAD_DIR.
- Improved the parsing of packrat.lock files to better represent the relationships between dependencies in the graph.
- The version of Detect is no longer part of the code location name.

Version 5.3.3

- Resolved an issue wherein reports for projects containing risks may be generated with a status of zero risks shown.

Version 5.3.2

- Synopsys Detect version 5.3.2 is a minor maintenance release.

Version 5.3.1

New features

- Added new property detect.ignore.connection.failures which enables Synopsys Detect to continue even if it fails to talk to Black Duck.

Resolved issues

- Resolved an issue wherein build scan failures may occur in TFS with the error `[COPY Operation] noSuchPath in source, path provided: //license/ownership`.
- Resolved an issue wherein if the property detect.clone.project.version.name is set to a non-existent project version, the log messages are now improved to make it easier to recognize the problem.

Changed features

- In cases where the property detect.clone.project.version.name is set to a non-existent project version, the log messages are now improved to make it easier to recognize the issue.

Version 5.2.0

New features

- Added support for Bazel.
- Added support for CMake.
- Added a property to support using project version nicknames.
- Added a property for application ID.
- Added Java wildcard pattern support.
- Added support for Coverity on Polaris.

Resolved issues

- Resolved an issue wherein the package-lock.json file may be missing additional versions.
- Resolved an issue wherein multiple simultaneous Detect executions may cause BDIO merges.
- Resolved an issue wherein permission errors may display when creating projects or scanning.

Changed features

- The --detect.bom.aggregate.name property now checks for an empty BOM. If the BOM is empty, it is not uploaded to Black Duck.
- Added support for PiP versions 6.0.0 and higher.
- Improved error messages for Black Duck connection issues.
- Cosmetic changes: from Black Duck Detect to Synopsys Detect.
- Streamlined execution of Coverity and Black Duck scans through a single continuous integration job.
- Updated location of the shell/PowerShell scripts.
- Updated location of the air-gapped archive.

Version 5.1.0

New features

- Added support for GoVendor.
- Added executable output to diagnostic mode.
- Added the project/version GUID in the console output.
- Added error codes.

Resolved issues

- Resolved an issue that fixes the Clang Detector (for C/C++) handling of complex quoted strings occurring in compiler commands found in the JSON compilation database (compile_commands.json) file.
- Resolved an issue wherein a Null Pointer Exception error may occur when Detect cannot access a file during signature scan exclusion calculating.
- Resolved an issue wherein the RubyGems package manager had missing components.
- Resolved an issue wherein the NPM package lock added every dependency as a root dependency.

Changed features

- The properties --detect.nuget.path and --detect.nuget.inspector.name are deprecated.
- The properties detect.suppress.results.output and detect.suppress.configuration.output are deprecated. The output from these properties is logged instead of written to sysout.
- Improved the reporting of scan registration limit errors.

Version 5.0.1

Resolved issues

- Resolved an issue wherein a null pointer exception error may occur in the NuGet portion of a scan when running Synopsys Detect in Linux.

- Resolved an issue that fixes the Clang Detector (for C/C++) handling of complex quoted strings occurring in compiler commands found in the JSON compilation database (compile_commands.json) file.
- Resolved an issue wherein using detect.tools=ALL did not run any tools.
- Resolved an issue wherein Coverity on Polaris may return a failure status for a successful upload.

Changed features

- NuGet air gap mode now points to other folders.
- Removed support for PiP resolving the project version.

Version 5.0.0

New features

- Added a new property to execute Black Duck Docker Inspector.
- CocoaPods are now nestable under Bill of Materials (BOM) tools.
- Added functionality to exclude all BOM tools.
- Added a new property which enables you to search at a determined depth.
- Added functionality to log all found executables.
- Added functionality to run in Docker mode.
- Added support for NuGet in MacOS.
- Added ability to include and exclude all tools.
- Added new properties for SWIP in Detect scans.

Resolved issues

- Resolved an issue that caused the Gradle inspector to retrieve the maven-metadata.xml file from the default repository, even when the property detect.gradle.inspector.repository.url was set to point to a different repository.
- Resolved an issue wherein Gradle may upload older BDIO files into the current project.

Changed features

- Improved C/C++ multi-threading functionality.

- Deprecated Pipenv inspector messages are now logged.
- The term BOM_TOOL is now replaced with DETECTOR.
- You can no longer supply ranges for the Inspector versions.
- Enhanced the code location naming conventions.

3

Requirements and release information

General requirements

- Normally, access to the internet is required to download and run Synopsys Detect and components from GitHub and other locations. For running without internet access, refer to *Air Gap Mode*.
- Minimum 8GB RAM.
- Java: OpenJDK 64-bit version 8, 11, 13, 14, 15, 16, or 17. If using Java 11: 11.0.5 or higher is required.
- curl versions 7.34.0 or later.
- Bash.
- If using detect8.ps1: PowerShell versions 4.0 or higher.
- The tools required to build your project source code.

Black Duck integration requirements

- Licensed installation of the current version of Black Duck with access credentials. Visit the *Black Duck release page* to determine the current version of Black Duck.
- For information about additional compatible versions of Black Duck, consult the *Black Duck Release Compatibility matrix.*
- The Black Duck notifications module must be enabled.
- A Black Duck user with the *required roles*.
- On Alpine Linux you will also need to override the Java installation used by the Black Duck Signature Scanner as described *here*.

Project type-specific requirements

In general, the detectors require:

- All dependencies must be resolvable. This generally means that each dependency has been installed using the package manager's cache, virtual environment, and others.
- The package manager / build tool must be installed and in the path.

Refer to the applicable *package manager sections* for information on specific detectors. This is particularly important for the *Docker Inspector* and the *NuGet Inspector*.

Risk report requirements

The risk report requires that the following fonts are installed:

- Helvetica
- Helvetica bold

Supported Synopsys Detect versions and Service duration

- For information about support and service durations for Synopsys Detect versions, consult the *Support and Service Schedule.*

4

Downloading and Installing Synopsys Detect

This section describes the process of downloading, verifying, and installing Synopsys Detect.

Refer to the menu topics for information about aquiring and installing the application in your environment.

Download Locations for Synopsys Detect & Plugins

The following are download locations for the current version of Synopsys Detect:

- Synopsys Detect Bash script: *Bash script*
- The Synopsys Detect PowerShell script: *PowerShell script*
- The Synopsys Detect binary repository (.jar and air gap zip files): *Binary files*
- The Synopsys Detect binary repository user interface view (properties, etc.): *Artifactory UI*
- The Synopsys Detect repository for the Jenkins plugin: *Artifactory*

Note: For certain types of projects, Synopsys Detect automatically downloads one or more *inspectors* as needed.

- In air-gap environments you may need to download the Sigma scanner via artifactory at the following location: *Sigma*

Upgrading Synopsys Detect

We recommend reading the release notes for each new Synopsys Detect version.

Synopsys Detect version names follow *semantic versioning*. Version strings follow the pattern MAJOR.MINOR.PATCH, with the following implications:

A PATCH version contains only fixes to functionality that already existed.

A MINOR version contans new features, and fixes to functionality that already existed.

Every MAJOR version (e.g. 7.0.0, 8.0.0, etc.) contains breaking changes. These breaking changes may not affect every user, but every user needs to check to see whether and how they to change the way they call Synopsys Detect before upgrading to the next MAJOR version. To do this check and and upgrade to the next MAJOR version: In a test environment:

1. Upgrade to the latest MINOR.PATCH Synopsys Detect version available for the MAJOR version you are currently running. Read all of the deprecation messages and the upgrade guidance they provide, and change the way you are calling Synopsys Detect until all deprecation messages are

gone. Read all of the documentation for the new properties you are using, and all of the documentation relevant to the features they control.

2. Upgrade to the next MAJOR version.
3. Test.

You must do this one MAJOR version at a time (do not skip over a MAJOR version).

For example, suppose you are running 7.12.1, and you want to upgrade to 8.0.0: In a test environment:

1. Upgrade to 7.14.0 (the latest 7.y.z version available). Read all of the deprecation messages and the upgrade guidance they provide, and change the way you are calling Synopsys Detect until all deprecation messages are gone. Read all of the documentation for the new properties you are using, and all of the documentation relevant to the features they control.
2. Upgrade to 8.0.0.
3. Test.

Synopsys Detect Version Management

Synopsys Detect self updating feature will allow customers who choose to enable Centralized Synopsys Detect Version Management in Black Duck to automate the update of Synopsys Detect across their pipelines.

Self updating Synopsys Detect scenarios

The Self Update feature will call the `/api/tools/detect` API to check for the existence of a mapped Synopsys Detect version in Black Duck. If a version that is eligible for upgrade or downgrade has been mapped, the API will redirect the request to download that version and the current execution of Synopsys Detect will invoke the downloaded version to execute the requested scan.

Synopsys Detect will download the required version from sig-repo or from a custom URL as configured in Black Duck. Centralized Synopsys Detect Version Management feature support in Black Duck is available from Black Duck version 2023.4.0 onwards.

Scenarios where Synopsys Detect self update will not execute

If there exists no mapping in Black Duck, or if the current version of Synopsys Detect matches the mapped version in Black Duck, or any issue occurs during the execution of the Self Update feature, then Synopsys Detect will continue with the current version to execute the scan.

If the Synopsys Detect URL of the Synopsys Detect .jar file to download and run has been hardcoded via Synopsys Detect property `DETECT_SOURCE` environment variable or the Synopsys Detect version set by the `DETECT_LATEST_RELEASE_VERSION` or `DETECT_VERSION_KEY`

variables, self update will not occur. These are optional System environment properties used by Detect upgrade scripts.

If the Black Duck “Internally Hosted” option has been selected and a Synopsys Detect download location has not been provided, the feature will not be enabled.

For further Black Duck configuration information, refer to the documentation provided under the topic: *Hosting location for Synopsys Detect.*

Synopsys Detect log examples for self update

Downgrade to prior version blocked:

```
2023-05-05 12:20:57 EDT INFO  \[main] - Detect-Self-Updater:  Checki
ng https://test1.synopsys .com/api/tools/detect API for centrally ma
naged Detect version to download to /Users/testuser/tmp.

2023-05-05 12:21:03 EDT WARN  \[main] - Detect-Self-Updater:  The De
tect version 8.7.0 mapped at Black Duck server is not eligible for d
owngrade as it lacks the self-update feature. The self-update featur
e is available from 8.9.0 onwards.
```

Update to version allowed (8.9.0+):

```
2023-05-05 12:33:52 EDT INFO  \[main] - Detect-Self-Updater:  Checki
ng https://test1.synopsys .com/api/tools/detect API for centrally ma
naged Detect version to download to /Users/testuser/tmp.

2023-05-05 12:33:53 EDT WARN  \[main] - Detect-Self-Updater:  The De
tect version 8.9.2 mapped at Black Duck server is eligible for downg
rade from the current version of 8.10.0. The self-update feature is
available from 8.9.0 onwards.

2023-05-05 12:33:53 EDT INFO  \[main] - Detect-Self-Updater:  Centra
lly managed version of Detect was downloaded successfully and is rea
dy to be run: /Users/testuser/tmp/synopsys-detect-8.9.2.jar.
```

Current version of Synopsys Detect matches the mapped version or there is no mapped version in Black Duck:

```
2023-05-05 12:33:52 EDT INFO  \[main] - Detect-Self-Updater:  Checki
ng https://test1.synopsys .com/api/tools/detect API for centrally ma
naged Detect version to download to /Users/testuser/tmp.

2023-05-05 12:33:53 EDT INFO  \[main] - Detect-Self-Updater:  Presen
t Detect installation is up to date - skipping download.
```

Important:

- Downgrading to versions earlier than 8.9.0 is not supported.
- This feature is not available in offline 'blackduck.offline.mode=true' or AirGap configurations or if the Black Duck URL has not been provided via the `blackduck.url` variable.
- When running an "Internally Hosted" instance of Synopsys Detect and using custom scripts, checks should be made to prevent Detect from querying Black Duck for version management and re-downloading itself.
- Self update makes it easy to switch to a new major Synopsys Detect version, so care should be taken to validate that automated scanning is not impacted.

Synopsys Detect Code Verification

Two methods are available to verify that the Synopsys Detect code you run has not been tampered with since it was built by Synopsys: code signature verification and checksum verification. Both methods apply to the Synopsys Detect .jar file, and only offer protection when you run Synopsys Detect by invoking the Synopsys Detect .jar file directly (as opposed to invoking detect8.sh or detect8.ps1).

Code signature verification

Code signature verification is the most secure method available for verifying Synopsys Detect code. This method relies on Java tools.

It involves verifying the Synopsys Detect .jar file that you download from the location specified in *download locations*, using the Java *jarsigner* tool. In the event that the .jar has been tampered with, verification will fail.

To verify the Synopsys Detect .jar:

jarsigner -verify -strict {your Synopsys Detect .jar file}

The output should be `jar verified.` (with no warnings).

Checksum verification

Checksum verification provides less protection against tampering than code signature verification provices because in the unlikely scenario the Synopsys SIG Artifactory server has been compromised, an attacker could alter both the .jar and the checksum. But checksum verification does provide some degree of protection against other attack scenarios.

The binary repository provides SHA-256, SHA-1, and MD5 checksums for each Synopsys Detect .jar file. To find it, navigate to the .jar file in the Synopsys SIG Artifactory server specified in *download locations*, and scroll to the bottom of the page. Various tools (such as md5sum, sha1sum, and sha256sum on Linux, and certutil and Get-FileHash on Windows) are available for calculating checksums of files on your computer. Use one of those tools to get a checksum for your copy of the Synopsys Detect .jar, and compare it to the corresponding checksum on the binary repository page to make sure they match.

Air Gap Mode

To run Synopsys Detect on an air-gapped computer or network, you must first download and install Synopsys Detect and dependencies that Synopsys Detect normally downloads as it runs. These include inspectors for Docker and NuGet, libraries that Gradle inspector requires, and other files. These files are packaged together in an air-gap archive that will be extracted on the target system.

Downloading or creating an air gap archive

Air gap archives are available for download from the location specified in *download locations*. These air gap archives contain the versions of the dependencies that were current at the time of the Synopsys Detect release.

As an alternative, you can create an air gap archive yourself. An air gap archive that you create will contain the versions of the dependencies that are current at the time you create the air gap archive (the same versions Synopsys Detect would download if run at that time).

To create an air gap archive, run Synopsys Detect with the -z or --zip command line option. Optionally you can follow --zip with a space and an argument (for example: --zip FULL) to customize the air gap zip. Possible values: FULL (produce a full air gap zip; the default), NO_DOCKER (do not include the Docker Inspector).

Your PATH environment variable must include the *bin* directory of the Gradle distribution to generate an Air Gap archive.

Running in air gap mode

For information refer to *Running in air gap mode*.

5

Quickstart guide

The following is a simple example to help you get started using Synopsys Detect.

Step 1: Locate or acquire a source code project on which you will run Synopsys Detect.

To run Synopsys Detect on junit4, which is an open source project written in Java and built with Maven, you could acquire it by doing the following:

```
git clone https://github.com/junit-team/junit4.git
cd junit4
```

To understand what Synopsys Detect does, it can be helpful to think about what you would do if you wanted to discover this project's dependencies without using Synopsys Detect. You might do the following:

1. Look in the project directory (junit4) for hints about how dependencies are managed. In this case, the *mvnw* and *pom.xml* files are hints that dependencies are managed using Maven.
2. Since it's a Maven project, you would likely run `./mvnw dependency:tree` to reveal the project's dependencies; both direct and transitive.

This is basically what Synopsys Detect does on this project. In addition, Synopsys Detect runs the Black Duck Signature Scanner on the directory, which may discover additional dependencies added to the project by means other than the package manager.

Step 2: Run Synopsys Detect connected to Black Duck.

To run Synopsys Detect, you will need to provide login credentials for your Black Duck server. One way to do that is to add the following arguments to the command line:

- `--blackduck.url={your Black Duck server URL}`
- `--blackduck.api.token={your Black Duck access token}`

The command you run looks like this:

On Linux or Mac:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --blackduck.u
rl={your Black Duck server URL} --blackduck.api.token={your Black Duck
access token}
```

On Windows:

```
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
blackduck.url={your Black Duck server URL} --blackduck.api.token={your
Black Duck access token}
```

The operations performed by Synopsys Detect depends on what it finds in your source directory. By default, Synopsys Detect considers the current working directory to be your source directory.

In the junit4 case, Synopsys Detect will:

1. Run the Maven detector, which uses Maven to discover dependencies.
2. Run the Black Duck Signature Scanner which scans the files in the source directory to discover dependencies.
3. Upload the discovered dependencies to Black Duck.
4. Provide in the log a Black Duck Project BOM URL that you can use to view the results in Black Duck.

Point your browser to the Black Duck Project BOM URL to see the Bill Of Materials for junit4.

Next steps

Synopsys Detect can be used on a variety of project types, and in a variety of ways, requiring its behavior to be highly configurable. For more detailed information on how to configure Synopsys Detect for your needs, see *Configuring Synopsys Detect*.

6

Getting started with Synopsys Detect

Before you start using Synopsys Detect, it is recommended that you become familiar with the makeup of Synopsys Detect and how it works. Review the 'Getting started with Synopsys Detect' pages listed in the left-hand menu, to learn about, and prepare to start using Synopsys Detect.

> Documentation for previous Synopsys Detect versions available via the *user guide archive*.

Key Concepts and Terms

This section describes the key concepts and terms that are used with Synopsys Detect. Understanding these terms and Synopsys Detect components will help you scan and analyze your code more efficiently and effectively.

Software Composition Analysis (SCA)

Open source software detection to provide users visibility into their open source inventory.

Synopsys Detect run

Typically, it consists of the Synopsys Detect detector using the project's package manager to derive the hierarchy of dependencies in a software project, running the Black Duck Signature Scanner, and uploading the results to Black Duck for analysis.

Synopsys Detect script

The primary function of the Synopsys Detect scripts is to download and execute the Synopsys Detect JAR file, which enables the scan.

You download and run the latest version of Synopsys Detect using the following commands, and add properties to refine the instruction.

Windows:

```
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect"
```

Linux/MacOs:

```
bash <(curl -s https://detect.synopsys.com/detect8.sh)
```

Synopsys Detect JAR

By using a specific Synopsys Detect JAR, you have direct control over the Synopsys Detect version that you use, rather than using the script, which automatically runs the latest version.

Synopsys Detect tools

Synopsys Detect tools are run to enable the scanning of your code.

The default tools that are run are:

- Detector (--detect.tools=DETECTOR). The detector tool runs the appropriate detectors that are used to find and extract dependencies by using package manager inspection.
- Black Duck Signature Scanner (--detect.tools=SIGNATURE_SCAN). The Black Duck Signature Scanner tool runs by default when Black Duck connection details are provided. A file/folder (Signature) scan is performed on the built project to examine all project files for open-source software.

Other Synopsys Detect tools such as Docker Inspector or Black Duck - Binary Analysis are not run by default in most scenarios but you can add them by using properties on the command line.

Detectors

Synopsys Detect uses detectors to find and extract dependencies from all supported package managers. For example, the Maven detector, which is run by default, executes an mvn dependency:tree command against a Maven project and derives dependency information, which can be sent to Black Duck

By default, all detectors are eligible to run. The set of detectors that actually run depends on the files that exist in your project directory.

Properties

A property to which you assign a value is like a flag or a parameter on the command line or in a script that provides instructions for the Synopsys Detect scan task.

When setting a property value, the property name is prefixed with two hyphens (--).

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) <--property=v
alue>
```

Example using properties to specify project name and Black Duck URL:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.name=MyProject --blackduck.url=https://blackduck.yourdomain.com
```

Inspectors

Inspectors are used by detectors when the package manager requires an integration or embedded plugin to work. For example, Gradle uses an inspector as a plugin that executes a custom task. Most detectors do not require an inspector.

Scans and projects

Synopsys Detect scans are mapped to one project. A project version can have more than one scan mapped to it, which enables the mapping of multiple separate folders and their scan results into one aggregated project version.

BDIO

Synopsys Detect produces dependency information for Black Duck, and additional Synopsys products and platforms, in Black Duck Input Output (BDIO) format files.

Vulnerability Impact Analysis

When the *detect.impact.analysis.enabled* property in Synopsys Detect to set to true, Synopsys Detect creates a call graph (a list of calls made by your code) to understand the public methods your code is using in your application. The call graph shows the fully qualified public method names as well as the line number where the function was called. The data is packaged into a single file and Synopsys Detect sends the file over HTTPS to the Black Duck server, enabling vulnerability impact analysis in Black Duck

How Synopsys Detect Works

This page provides an overview of how Synopsys Detect works.

How Synopsys Detect does its work

Synopsys Detect performs the following basic steps when scanning open source software, assuming you are connected to a Black Duck instance.

1. Synopsys Detect uses the project's package manager to derive the hierarchy of dependencies known to that package manager. For example, on a Maven project, Synopsys Detect executes an mvn dependency:tree command, and derives dependency information from the output.
2. Runs the Black Duck Signature Scanner on the project. This might identify additional dependencies not known to the package manager such as a .jar file copied into the project directory.

3. Uploads both sets of results (dependency details) to Black Duck creating the project/version if it does not already exist. Black Duck uses the uploaded dependency information to build the Bill Of Materials (BOM) for the project/version.

In this case, the user has provided Black Duck connection details through property settings to Synopsys Detect, specifying that results (project dependency details) are to be uploaded to Black Duck By combining all these techniques, Synopsys Detect is capable of scanning a wide range of software projects utilizing a variety of package managers and programming languages for open source components.

Synopsys Detect Basic Workflow

Using Synopsys Detect to analyze your code typically involves the following basic steps:

1. Satisfy system and other execution requirements
2. Download Synopsys Detect
3. Determine how you will configure Synopsys Detect to connect to your Black Duck server
4. Determine how you will configure Synopsys Detect to execute the desired functions
5. Run Synopsys Detect
6. Examine the analysis results
7. Generate desired reports

Synopsys Detect Processing

Synopsys Detect processing can be broken into the following phases:

Initialization phase

In this phase, Synopsys Detect does verification checks on the user-provided configuration, checks connectivity to any external systems needed for the run, and creates any required directories.

Run phase

In this phase, Synopsys Detect processes an ordered list of tools, invoking all that apply, which depends on how Synopsys Detect is configured.

Synopsys Detect analysis is done using an ordered set of tools that you specify using Synopsys Detect properties.

- By default, the build detector tool is run. This detector runs after a build and has access to both build artifacts and build tools; it produces the most accurate results.

- If Black Duck connection details are provided, the Black Duck signature scanner tool also runs by default.

Depending on project contents, the detector tool runs different types of detectors to find and extract dependencies from supported package managers. For example, if Synopsys Detect finds a pom.xml file, it runs the Maven detector. If Synopsys Detect finds Gradle files, it runs the Gradle detector.

At the end of the run phase, Synopsys Detect uploads results to Black Duck, and optionally performs tasks such as generating a risk report or checking for policy violations.

Cleanup phase

During the cleanup phase, Synopsys Detect removes temporary files and directories before exiting.

Configuration Overview

What Synopsys Detect looks at and how it performs its analysis depends on how you configure Synopsys Detect.

> For accurate SCA analysis, Synopsys Detect should be executed as a post-build step typically in the native build environment.

Using properties, you can configure the following:

- What code to examine
- Authentication information
- Connection information
- Required detectors
- Sensitivity to policy violations
- Reporting
- Logging

User role requirements when running with Black Duck

Any user can download Synopsys Detect and run a scan, however you must configure a user/API token in Black Duck for the Synopsys Detect scan to be analyzed by Black Duck.

For more information on creating a Black Duck user token, please consult the documentation provided by Black Duck under the topic: *Managing user access tokens.*

The following user roles are required for the user that you create in Black Duck

- The user must have the Project Creator overall role in order to create Black Duck projects.

- The user must have the Global Project Viewer overall role, or be a member of the project, in order to create Black Duck project versions.
- The user must have the Project Code Scanner project role, or the Global Code Scanner overall role, in order to populate the project BOM.

Command line help options

Synopsys Detect provides the following execution options to assist with configuration, execution and support.

Mode	Command line option	Alt. option	Description
Help	--help	-h	Provides basic help information (including how to get more detailed help).
Interactive	--interactive	-i	Guides you through configuring Synopsys Detect.
Diagnostic	--diagnostic	-d	Creates a zip file containing diagnostic information for support.
hyaml	--helpyaml	-hyaml	To generate a configuration file template for setting properties.

Additional resources are available at *Synopsys Software Integrity Community*.

7

Configuring Synopsys Detect

Synopsys Detect is configured by assigning values to properties. See the left-hand menu for further configuration means and methods.

Synopsys Detect's configuration mechanisms are provided by Spring Boot. Additional details on configuring Spring Boot applications like Synopsys Detect can be found in the *Spring Boot documentation.*

On the command line

One method for configuring Synopsys Detect is by setting *property values* on the command line. When setting a property value on the command line, prefix the property name with two hyphens (--).

To add one property setting to the command line, add the following at the end:

```
{space}--{property name}={value}
```

There is a space before and between each complete property setting, but there are no spaces around the equals sign (=).

For example, to set property *detect.project.name*:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.name=MyProject
```

Using environment variables

Synopsys Detect properties can also be set using environment variables.

On Linux, when setting a property value using an environment variable, the environment variable name is the property name converted to uppercase, with period characters (".") converted to underscore characters ("_"). For example:

```
export DETECT_PROJECT_NAME=MyProject
bash <(curl -s -L https://detect.synopsys.com/detect8.sh)
```

On Windows, the environment variable name can either be the original property name, or the property name converted to uppercase with period characters (".") converted to underscore characters ("_"). For example:

```
$Env:DETECT_PROJECT_NAME = MyProject
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect"
```

Using a configuration file

Another commonly-used method of configuring Synopsys Detect is to provide a configuration file. The configuration file can be a Java properties (.properties) file, or a YAML (.yml) file.

Spring Boot will look for a configuration file named application.properties or application.yml in the current working directory, or a ./config subdirectory. If it exists, it will read property values from it.

For example, if you wanted to set property *detect.project.name* using a configuration (.properties) file, you could do it as follows:

```
echo "detect.project.name=myproject" > application.properties
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.sour
ce.path=/opt/projects/project1
```

Because the configuration file has one of the file names that Spring looks for by default (in this case, application.properties) and exists in one of the locations that Spring looks in by default (in this case, the current directory), there is no need to specify the path to the configuration file on the command line.

Additional details can be found in the *Spring Boot documentation*.

Properties file

When setting a property value in a .properties file, do not prefix the property name with hyphens, and adhere to Java .properties file syntax: `propertyName=propertyValue`, one per line.

YAML file

When setting a property value in a .yml file, do not prefix the property name with hyphens, and adhere to YAML syntax for dictionaries: `propertyName: propertyValue`, one per line. There is a Synopsys Detect command line help option, -hyaml, that can be used to generate a template YAML configuration file.

Running Synopsys Detect from a directory that contains a file named *config*

If a file named *config* exists in the directory from which you run Synopsys Detect, you must override the default value of the Spring Boot property *spring.config.location* so that Spring Boot does not try to read that file as a directory. If you are using a Spring Boot configuration file such as application.properties or application.yml, set the value of *spring.config.location* so Spring Boot will find your configration file. That is, set it to the path of the directory in which that file resides (include a trailing slash to indicate that you are specifying a directory), or to the path of the config file itself.

If you are not using a Spring Boot configuration file, set the value of *spring.config.location* to the empty string:

```
--spring.config.location=""
```

Switching between multiple profiles

A profile is, in effect, a set of pre-defined properties. You select the profile (property settings) you want when you run Synopsys Detect.

Creating a profile

To define a set of properties for a profile, create a configuration file named *application-{profilename}.properties* or *application-{profilename}.yml* in the current working directory, or in a subdirectory named *config*. Populate it with property assignments as previously described.

Selecting a profile on the command line

To select one or more profiles on the Synopsys Detect command line, assign the the comma-separated list of profiles to the Spring Boot property *spring.profiles.active*:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --spring.prof
iles.active={profilename}
```

This capability is provided by Spring Boot. For more information, refer to *Spring Boot's profile mechanism*.

Additional configuration methods and details

Synopsys Detect reads property values using *Spring Boot's externalized configuration mechanism*, which provides capabilities beyond those described on this page.

The most common methods used to pass a property value to Synopsys Detect are listed as follows. A method with a lower number in Spring Boot's order of precedence overrides a method with a higher number.

- Using a command line argument, (#4 in Spring Boot's order of precedence):

```
--blackduck.url=https://blackduck.yourdomain.com
```

- Using one environment variable per property, (#10 in Spring Boot's order of precedence):

```
export BLACKDUCK_URL=https://blackduck.yourdomain.com
```

- Using property assignments in a .properties configuration file, (#14 in Spring Boot's order of precedence):

```
blackduck.url=https://blackduck.yourdomain.com
blackduck.api.token=youraccesstoken
```

- Using property assignments in a .yml configuration file, (#14 in Spring Boot's order of precedence, however .properties takes precedence over .yml):

```
blackduck.url: https://blackduck.yourdomain.com
blackduck.api.token: youraccesstoken
```

- Using the SPRING_APPLICATION_JSON environment variable with a set of properties set using JSON format, (#5 in Spring Boot's order of precedence):

```
export SPRING_APPLICATION_JSON='{"blackduck.url":"https://blackduck.you
rdomain.com","blackduck.api.token":"youraccesstoken"}'
```

- Cross referencing a system property as a value in the Spring Boot JSON property, (#10 in Spring Boot's order of precedence):

```
export SPRING_APPLICATION_JSON='{"blackduck.url":"${BLACKDUCK_URL}"}'
```

Refer to the *Spring Boot documentation* for more details and advanced ways to set properties.

Providing sensitive values such as credentials

You can provide sensitive values such as credentials to Synopsys Detect using a variety of mechanisms provided by *Spring Boot*, including:

- On the command line; for example, --blackduck.api.token={your access token}.
- As an environment variable value; for example, export BLACKDUCK_API_TOKEN={your access token}.
- In a configuration (.properties) file; for example, ./application.properties.

Values provided on the command line may be visible to other users that can view process details. Setting sensitive values using environment variables is usually considered more secure.

Path properties

Each Synopsys Detect property with a type of "Path" or "Optional Path" accepts a file path value. (These properties also tend to have names that end with ".path".) The file path value can be either absolute (e.g. /usr/bin/conan) or relative to the directory from which Synopsys Detect is executed (e.g. ../bin/conan). The format of the file path (the directory separator character, whether or not a drive letter prefix is supported, etc.) is dictated by the operating system on which Synopsys Detect is running.

Path property value examples (applicable when executing the Synopsys Detect .jar directly):

- Linux/Mac: --detect.conan.path="/usr/bin/conan"
- Windows: --detect.npm.path="C:\Program Files\nodejs\npm.cmd"

When running Synopsys Detect using one of the scripts, remember to also apply quoting and escaping rules that apply. For more information refer to *Quoting and escaping shell script arguments*.

Property wildcard support

The values of the following Synopsys Detect properties can utilize filename globbing-style wildcards described below:

- detect.maven.included.scopes
- detect.maven.excluded.scopes
- detect.maven.included.modules

- detect.maven.excluded.modules
- detect.gradle.included.configurations
- detect.gradle.excluded.configurations
- detect.gradle.included.projects
- detect.gradle.excluded.projects
- detect.binary.scan.file.name.patterns
- detect.lerna.included.packages
- detect.lerna.excluded.packages
- detect.excluded.directories

The supported wildcards and their effect are:

- An asterisk (*) matches any sequence of zero or more characters
- A question mark (?) matches any single character

For example:

- *.jpg matches someimage.jpg, but not somedocument.doc
- *.??? matches someimage.jpg and somedocument.doc, but not somedocument.docx

Wildcard evaluation in these values is similar to Linux command line file globbing, and different from regular expression matching.

Synopsys Detect uses the *Apache Commons IO FilenameUtils.wildcardMatch()* method to determine whether a string matches the given pattern.

Java regular expression support

The values of the following Synopsys Detect property can utilize Java regular expressions described below:

- blackduck.proxy.ignored.hosts

A few of the supported wildcards and their effect are:

- A period (.) matches any single character
- An asterisk (*) matches a sequence of zero or more of the preceding character
- A backslash (\) preceding a special regular expression character (such as . or *) causes that character to be treated as the literal character

For example:

- .*-proxy01\.dc1\.lan matches qa-ssl-proxy01.dc1.lan and prod-proxy01.dc1.lan, but not qa-ssl-proxy02.dc1.lan
- qa-ssl-proxy..\.dc1\.lan matches qa-ssl-proxy01.dc1.lan and qa-ssl-proxy02.dc1.lan

Synopsys Detect uses the *Java Pattern class* to determine whether a string matches the given pattern.

Shell script configuration

The Synopsys Detect Bash and PowerShell scripts are configured by setting the environment variables.

In Bash, an environment variable is set as follows:

```
export ENV_VAR_NAME=value
```

In Command or Batch, an environment variable is set as follows:

```
set ENV_VAR_NAME=value
```

In PowerShell, an environment variable is set as follows:

```
$Env:ENV_VAR_NAME = value
```

It is generally a good idea to quote the value as you assign it. Be sure to check that the variable value has been set as expected by displaying its value after you have set it.

Refer to *Running the Synopsys Detect script* for more information.

Variable	Purpose	Value	Notes
DETECT_SOURCE	Download the Synopsys Detect .jar from a given URL	The URL of the Synopsys Detect .jar file to download and run	

Variable	Purpose	Value	Notes
DETECT_LATEST_RELEASE_VERSION	Run a given Synopsys Detect version	A Synopsys Detect version (example: 5.6.2)	If you would like to run a Synopsys Detect version other than the latest, set DETECT_LATEST_RELEASE_VERSION to the Synopsys Detect version you would like to run (for example: 5.6.2). DETECT_SOURCE has precedence over DETECT_LATEST_RELEASE_VERSION. You can see the available Synopsys Detect versions in the binary repository specified in *download locations*.
DETECT_VERSION_KEY	Continue running an earlier major version of Synopsys Detect	DETECT_LATEST (default), DETECT_LATEST_5, DETECT_LATEST_4	If neither DETECT_SOURCE nor DETECT_LATEST_RELEASE_VERSION is specified, the script will use the version key to query Artifactory for the correct version to download. By default it will look for DETECT_LATEST, however the Synopsys Detect artifactory also includes keys for some of the major versions of Synopsys Detect such as DETECT_LATEST_4. You can view the available values for DETECT_VERSION_KEY in Synopsys Detect project in the binary repository specified in *download locations*.

Variable	Purpose	Value	Notes
DETECT_JAR_PATH	Change the Synopsys Detect .jar download dir	The path to the .jar file download directory	If DETECT_JAR_PATH is provided, the script will use this location when downloading and running detect. The location of the jar will be DETECT_JAR_PATH/synopsys-detect-{version}.jar. The Bash script will default to '{user home directory}/synopsys-detect/download' if no option is specified.
TMP (PowerShell only)	Provides the user's temporary directory	The path to a directory for temporary files	If DETECT_JAR_PATH is not provided, the script will use the environment 'TMP' variable as the folder for the Synopsys Detect .jar path.
HOME (PowerShell only)	Provides the user's home directory	The path to the user's home directory	If DETECT_JAR_PATH is not provided and no 'TMP' variable can be found, the '$HOME/tmp' folder will be used for the Synopsys Detect jar path.
DETECT_JAVA_PATH	The path to the Java instance used to execute Docker Inspector	Path to the Java executable file.	To set the Java instance used by Synopsys Detect, invoke Detect using a specific Java executible or set JAVA_HOME.
JAVA_HOME	The path to the Java installation directory.	The path to the Java home directory.	If DETECT_JAVA_PATH is not set, and JAVA_HOME is set, the script will execute $JAVA_HOME/bin/java.

Variable	Purpose	Value	Notes
PATH	The executable program path.	The list of directories in which the system looks for the executable file for each command executed (the syntax is operating system-specific).	If neither DETECT_JAVA_PATH nor JAVA_HOME are set, the script assumes the directory containing the Java executable file is on the path.
DETECT_EXIT_CODE_PASSTHRU (PowerShell only)	Prevent the shell from exiting on completion	1	Setting this variable to '1' will cause the script to simply return the exit code but not exit. By default, the Synopsys Detect PowerShell script will exit with the exit code of Synopsys Detect. This is desirable because many CI's such as Team Foundation Server(TFS), will look at the scripts exit code to decide build status. It may be undesirable to exit the script in some situations such as when debugging in a terminal.
DETECT_SKIP_JAVA_TEST (PowerShell only)	Skip the test for the presence of the Java command	1	Setting this variable to '1' causes the script not to ensure that Java is on the path. By default the script will attempt to execute "java -version" to ensure that Java is available and executable.

Variable	Purpose	Value	Notes
DETECT_CURL_OPTS (Bash only)	Add the given options to any curl commands executed	curl command options (a string)	Use this variable to add options to the curl command used to download files such as the Synopsys Detect .jar file. For example, you can use this variable to set proxy settings for curl. The PowerShell script does not support this as it does not use curl. To supply proxy information to the PowerShell you can simply set the Synopsys Detect proxy settings as environment variables.
DETECT_JAVA_OPTS (Bash only)	Add the given options to the Java command	Java command options (a string)	Use this variable to add options to the Java command used to execute Synopsys Detect. The PowerShell script does not currently support this setting.
DETECT_DOWNLOAD_ONLY (Bash only)	Download the Synopsys Detect .jar file, but do not run it	1	Set this variable to 1 to download, but not run, the Synopsys Detect .jar file. The PowerShell script does not currently support this setting.
BLACKDUCK_PROXY_HOST (PowerShell only)	Proxy host to use to download detect		When set, the PowerShell script will use the configured proxy information to download detect. Supports both environment variable styles (see below).

Variable	Purpose	Value	Notes
BLACKDUCK_PROXY_PORT (PowerShell only)	Proxy port to use to download detect		When set, the PowerShell script will use the configured proxy information to download detect. Supports both environment variable styles (see below).
BLACKDUCK_PROXY_USERNAME (PowerShell only)	Proxy username to use to download detect		When set, the PowerShell script will use the configured proxy information to download detect. Supports both environment variable styles (see below).
BLACKDUCK_PROXY_PASSWORD (PowerShell only)	Proxy password to use to download detect		When set, the PowerShell script will use the configured proxy information to download detect. Supports both environment variable styles (see below).

Note that proxy environment variables can be provided to the PowerShell script in both 'Bash' and 'PowerShell' formats. For example a given property name can be specified as "PROPERTY_NAME" or "property.name". Note that the proxy can only be provided to the PowerShell script as an environment variable.

Quoting and escaping shell script arguments

Tip: Escaping characters via the command line can be complicated, to simplify any escaping requirements we recommend you consider using either *environment variable* or *configuration files*.

Running the Bash script (detect8.sh) on Linux or Mac

The recommended environment ("parent shell") for running detect8.sh on Linux is Bash, and Bash or Zsh on Mac.

Note: Exact requirements for escaping characters will depend on the set of properties and arguments provided. The following section provides general guidance.

When an argument contains a space or other non double quote special character, you can wrap the argument in single quotes, or escape the special character with a backslash (\). The quotes can surround either the value or the entire argument.

For example:

```
# name: Project Test
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.name='Project Test'

# name: Project Test
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.name=Project\ Test

# name: Project!Test
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.name=Project\!Test
```

You can include a double quote by single quoting the string, and escaping the double quotes with backslashes:

```
# license: BSD 3-clause "New" or "Revised" License
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.proj
ect.version.license='BSD 3-clause \"New\" or \"Revised\" License'
```

Running in Command Prompt (cmd) on Windows (detect8.ps1)

On Windows, you can run detect8.ps1 in either a *Windows Command Prompt* session, or a PowerShell session. This section describes running detect8.ps1 in a *Windows Command Prompt* session.

Note: Exact requirements for escaping characters will depend on the set of properties and arguments provided. The following section provides general guidance.

When an argument contains a space or other non quote special character, you can wrap the argument in single quotes, or escape the special character with a backtick (`). The quotes can surround either the value or the entire argument.

For example:

```
# name: Project Test
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
detect.project.name='Project Test'

# name: Project Test
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" '-
-detect.project.name=Project Test'

# name: Project Test
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
detect.project.name=Project` Test
```

When an argument contains a comma, you can wrap the argument in single quotes, and escape the special character with a backtick (`). In the case of a name with a comma and a space, you would use a backtick in front of both the comma and space.

For example:

```
# name: Project,Test
Powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
 https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect --
blackduck.url=<url> --detect.project.name='Project,Test'"

# name: Project,Test
Powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
 https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect --
blackduck.url=<url> '--detect.project.name=Project,Test'"

# name: Project, Test
```

```
Powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
 https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect --
blackduck.url=<url> --detect.project.name=Project`,` Test"
```

You can include a single quote by doubling it:

```
# name: singlequote'
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
detect.project.name='singlequote'''
```

You can include a double quote using this sequence: double quote, 2 backslashes, 2 double quotes:

```
# license: BSD 3-clause "New" or "Revised" License
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
detect.project.version.license='BSD 3-clause "\\""New"\\"" or "\\""Revi
sed"\\"" License'
```

Running in PowerShell on Windows (detect8.ps1)

An alternative environment for running detect8.ps1 on Windows is to run it from inside a PowerShell session.

This invocation has an important distinction from the Command Prompt invocation in that the script does NOT EXIT the session. This is desirable when running the script from a terminal session but not from within a CI/CD environment.

Tip: When running from within a CI/CD environment either omit the passthrough flag or use the command prompt invocation as the job may not set the proper exit code if the session does not exit.

Note: Exact requirements for escaping characters will depend on the set of properties and arguments provided. The following section provides general guidance.

When an argument contains a space or other non-quote special character, you can wrap the argument in single quotes, or escape the special character with a backtick. The quotes can surround either the value or the entire argument.

For example:

```
# name: Project Test
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
dom) | iex; detect --detect.project.name='Project Test'

# name: Project Test
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
dom) | iex; detect "--detect.project.name=Project Test"
```

When an argument contains a comma, you must escape the special character with a backtick (`). In the case of a name with a comma and a space, you would use a backtick in front of both the comma and space.

For example:

```
# name: Project,Test
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https:/
/detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect --detect.p
roject.name=Project`,Test

# name: Project, Test
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https:/
/detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect --detect.p
roject.name=Project`,` Test
```

You can include a double quote using this sequence: backslash, backtick, double quote:

```
# license: BSD 3-clause "New" or "Revised" License
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
```

```
dom) | iex; detect --detect.project.version.license="BSD 3-clause \`"Ne
w\`" or \`"Revised\`" License"
```

Project, Version, and Code Location Naming

The following sections describe the project, version, and code location (scan) naming in Synopsys Detect.

Project and version naming

The project and version names of the project to which Synopsys Detect writes results are, by default, derived from the project on which Synopsys Detect is run. The mechanism Synopsys Detect uses to determine the project and version names depends on project type. If Synopsys Detect cannot determine the project and version names, then Synopsys Detect uses the project directory name as the project name, and the value "Default Detect Version" as the version name.

You can use the following properties to override the project and version names:

```
--detect.project.name=PROJECT-NAME
--detect.project.version.name=VERSION-NAME
```

You can use the following property to change the default version to a timestamp:

```
--detect.default.project.version.scheme=timestamp
```

You can use the following property to customize the timestamp format:

```
--detect.default.project.version.timeformat='yyyy-MM-dd:HH:mm:ss.SSS'
```

Project and version naming for Git projects

If no package manager provides project and version names, you have not provided the project and version names through properties, and the project uses Git, Synopsys Detect attempts to use Git to determine project information.

Project information is extracted from the remote URL for the current branch. The version is the current branch name, or the commit hash if a detached head is checked out. This is done by the Git detector. If you don't want Synopsys Detect to use Git data, omit the Git detector using the following property:

```
--detect.excluded.detector.types=GIT
```

For example, for a project with a remote URL of "https://github.com/blackducksoftware/synopsys-detect" and a checked-out branch of "5.5.0", Synopsys Detect by default uses the project name "blackducksoftware/synopsys-detect" and project version "5.5.0".

Synopsys Detect attempts to derive project and version information by running the Git executable. If that is not successful, it attempts to derive project and version information by parsing Git files.

In Synopsys Detect versions 5.5.0 and higher, there is a new Synopsys Detect property for providing the path to the Git executable: detect.git.path.

Code location (scan) naming

Synopsys Detect often generates multiple code locations (scans) in a single run. Each code location name consists of a base name and a type suffix that indicates what type of scan generated it.

Scan type	Default base name	Suffix
Package manager (all detectors plus Docker Inspector and Bazel)	project/version	bdio
Impact analysis	directory/project/version	impact
Signature	directory/project/version	signature
Binary	file/project/version	binary
IaC	directory/project/version	iac

You can modify the base name by adding a prefix and/or a suffix to the default base name using the `detect.project.codelocation.prefix` and `detect.project.codelocation.suffix` properties.

You also have the option to set the base name using the `detect.code.location.name` property. When `detect.code.location.name` is set, `detect.project.codelocation.prefix` and `detect.project.codelocation.suffix` are ignored.

BDIO aggregation

Starting with version 8.0.0, Synopsys Detect aggregates all package manager results into a single BDIO file / codelocation.

All dependency graphs produced by any of the following, executed during the Synopsys Detect run, will be aggregated:

- Detectors

- Docker Inspector
- Bazel

This BDIO takes advantage of functionality added to Black Duck in version 2021.8.0 enabling Black Duck to preserve both source information (indicating, for example, from which subproject a dependency originated) and match type information (direct vs. transitive dependencies).

Synopsys Detect now operates in a way that is similar to Synopsys Detect 7 run with property detect.bom.aggregate.remediation.mode=SUBPROJECT. The property detect.bom.aggregate.remediation.mode does not exist in Synopsys Detect 8.

Related properties

- *detect.bdio.output.path*
- *detect.bdio.file.name*

8

Running Synopsys Detect

This section describes the basics of running Synopsys Detect.

Refer to the menu topics for information about planning, and running the application in your environment.

Planning

This section covers several planning and options to review before running Synopsys Detect.

Deciding how to use Synopsys Detect

Before you download and run Synopsys Detect, you need to make the following decisions:

- Do you want to run Synopsys Detect before you build or after?
- In which directory do you want to run Synopsys Detect?
- Do you want to run Synopsys Detect as a script or a .jar file; this affects which version is run.
- What *tools and detectors* do you want to include or exclude?
- Do you want to run Synopsys Detect offline, or connected to Black Duck.

Positioning Synopsys Detect in the build process

For best results, execute Synopsys Detect post-build step in the build environment of the project. Building your project prior to running Synopsys Detect is required for many detectors to run successfully, tends to produce the most accurate results, and helps ensure that the build artifacts are available for signature scanning.

When higher accuracy detectors are unable to run (due to, for example, the absence of package manager executables they need), and a lower accuracy detector is also available, Synopsys Detect makes its best effort to discover dependencies by running the lower accuracy detector.

See *Detector search and accuracy* for more information.

Choosing the working directory

You can run Synopsys Detect from any directory. If you are not running Synopsys Detect from the project directory, provide the project directory using the *source path property*. When that property is not set, Synopsys Detect assumes the current working directory is the project directory.

Choosing a run method (script, .jar, or Docker container)

There are three ways to run Synopsys Detect:

1. Download and run a Synopsys Detect *script*.
2. Download and run a Synopsys Detect *.jar file*.
3. Run Synopsys Detect from *within a Docker container*.

The primary reason to run one of the Synopsys Detect scripts is that the scripts have an auto-update feature. By default, they always run the latest version of the Synopsys Detect .jar file within a specific major version; downloading it for you if necessary. When you run Synopsys Detect via one of the provided scripts, you automatically pick up fixes and new features as they are released. Each script limits itself to a specific Synopsys Detect major version (for example, 7.y.z, or 6.y.z), unless you override this default behavior.

Synopsys Detect version	Script Type	Script Name
8	Bash	detect8.sh
8	PowerShell	detect8.ps1
7	Bash	detect7.sh
7	PowerShell	detect7.ps1

Instuctions and examples in this documentation that reference the scripts assume you are running Synopsys Detect 8, so refer to detect8.sh or detect8.ps1. To run Synopsys Detect 7 instead, substitute detect7.sh for detect8.sh, or detect7.ps1 for detect8.ps1.

The primary reason to run the Synopsys Detect .jar directly is that this method provides direct control over the exact Synopsys Detect version; Synopsys Detect does not automatically update in this scenario.

The primary reason to run Synopsys Detect from within a Docker container is to take advantage of the benefits of Docker containers, which include standardized run environment configuration; Synopsys Detect does not automatically update in this scenario.

Installation Best Practices

Manually installing Synopsys Detect enables you to assure that the running version is compatible with your environment. Invoking Synopsys Detect with the bash and powershell scripts is easy but automatically downloaded updates may not be compatible with your environment.

The best practice for resilience is to add Synopsys Detect on the path, allowing for an easier invocation than even the bash and powershell scripts. It still allows easy updating without modifying commands just as the bash and powershell scripts do. This is the recommended best practice approach when resiliency is required.

Basic Manual Installation Steps

1. Download Java and make sure it is on your PATH
2. Download the version of Synopsys Detect you want to use from https://sig-repo.synopsys.com/bds-integrations-release/com/synopsys/integration/synopsys-detect/
   - You should download the air-gap zip if you do not want Synopsys Detect to download Inspectors at runtime
3. Put the Synopsys Detect jar/zip somewhere you can manage it
   - Examples:
   - Mac/Linux: $HOME/synopsys-detect/download/synopsys-detect-X.X.X.jar
   - Windows: C:\Program Files\synopsys-detect\download\synopsys-detect-X.X.X.jar
4. You can now run Synopsys Detect
   - Example: java -jar $HOME/synopsys-detect/download/synopsys-detect-X.X.X.jar --help

Mac/Linux Best Practice Installation Steps for Resilience

1. Download Java and make sure it is on your PATH
2. Download the version of Synopsys Detect you want to use from https://sig-repo.synopsys.com/bds-integrations-release/com/synopsys/integration/synopsys-detect/
   - You should download the air-gap zip if you do not want Synopsys Detect to download Inspectors at runtime
3. Create a symlink for the Synopsys Detect jar
   - ```
     ln -s $HOME/synopsys-detect/download/synopsys-detect-X.X.X.jar $HOME/synopsys-detect/download/latest-detect.jar
     ```
4. Create a bash script named "detect" with the following content.
   - ```
     #!/bin/bash
     ```
   - ```
     java -jar $HOME/synopsys-detect/download/latest-detect.jar "$@"
     ```
5. Add the script to your PATH variable
     ```

- ```
  export PATH=${PATH}:${path_to_folder_containing_detect_script}
  ```

6. OR instead of altering your PATH you can place the script in a directory that is already on your PATH
   - Example: /usr/local/bin
7. You can now run Synopsys Detect
   - Example: detect --help

Windows Best Practice Installation Steps for Resilience

1. Download Java and make sure it is on your PATH
2. Download the version of Synopsys Detect you want to use from https://sig-repo.synopsys.com/bds-integrations-release/com/synopsys/integration/synopsys-detect/
   - You should download the air-gap zip if you do not want Synopsys Detect to download Inspectors at runtime
3. Create a symbolic link for the Synopsys Detect jar, called latest-detect.jar
   - Start a command prompt in the folder you downloaded detect.
   - Run the following: mklink latest-detect.jar synopsys-detect-X.X.X.jar
4. Create a bat script named "detect.cmd" in the same folder with the following content
   - ```
     @java -jar "C:\Program Files\synopsys-detect\download\latest-detect.jar" %*
     ```
5. Add the script to your PATH variable
   - In File Explorer right-click on the This PC (or Computer) icon, then click Properties -> Advanced System Settings -> Environment Variables
   - Under System Variables select Path, then click Edit
   - Add an entry with the path to the folder containing the script "C:\Program Files\synopsys-detect\download"
6. You can now run Synopsys Detect
   - Example: detect --help

Running the Synopsys Detect script

The primary function of the Synopsys Detect scripts is to download and execute the Synopsys Detect .jar file. Several aspects of script functionality can be configured, including:

- The Synopsys Detect version to download/run; by default, the latest version.
- The download location.
- Where to find Java.

Information on how to configure the scripts is in *Shell script configuration*.

Running the script on Linux or Mac

On Linux or Mac, execute the Synopsys Detect script (detect8.sh, which is a Bash script) from Bash.

To download and run the latest version of Synopsys Detect in a single command:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh)
```

Append any command line arguments to the end, separated by spaces. For example:

```
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --blackduck.u
rl=https://blackduck.mydomain.com --blackduck.api.token=myaccesstoken
```

See *Quoting and escaping shell script arguments* for details about quoting and escaping arguments.

To run a specific version of Synopsys Detect:

```
export DETECT_LATEST_RELEASE_VERSION={Synopsys Detect version}
bash <(curl -s -L https://detect.synopsys.com/detect8.sh)
```

For example, to run Synopsys Detect version 7.13.2:

```
export DETECT_LATEST_RELEASE_VERSION=7.13.2
bash <(curl -s -L https://detect.synopsys.com/detect8.sh)
```

Running the script on Windows

On Windows, you can execute the Synopsys Detect script (detect8.ps1, which is a PowerShell script),from *Command Prompt* or from inside a PowerShell session.

Running from Windows Command Prompt

To download and run the latest version of Synopsys Detect in a single command from Command Prompt:

```
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect"
```

Append any command line arguments to the end, separated by spaces. For example:

```
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect" --
blackduck.url=https://blackduck.mydomain.com --blackduck.api.token=myac
cesstoken
```

See *Quoting and escaping shell script arguments* for details about quoting and escaping arguments.

To run a specific version of Synopsys Detect from Command Prompt:

```
set DETECT_LATEST_RELEASE_VERSION={Synopsys Detect version}
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect"
```

For example, to run Synopsys Detect version 5.5.0:

```
set DETECT_LATEST_RELEASE_VERSION=5.5.0
powershell "[Net.ServicePointManager]::SecurityProtocol = 'tls12'; irm
https://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect"
```

Running from Windows Powershell

To download and run the latest version of Synopsys Detect in a single command from PowerShell:

```
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
dom) | iex; detect
```

Note that when running the above command, the PowerShell session is not exited. See here for more information on the difference between the two commands.

Append any command line arguments to the end, separated by spaces.

See *Quoting and escaping shell script arguments* for details about quoting and escaping arguments.

To run a specific version of Synopsys Detect from Powershell:

```
$Env:DETECT_LATEST_RELEASE_VERSION = "{Synopsys Detect version}"
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
dom) | iex; detect
```

Or:

```
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; $Env:DETECT_LATEST_RELEASE_VERSION = "{Synopsys Detec
t version}"; irm https://detect.synopsys.com/detect8.ps1?$(Get-Random)
| iex; detect
```

For example, to run Synopsys Detect version 7.0.0:

```
$Env:DETECT_LATEST_RELEASE_VERSION = "7.0.0"
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; irm https://detect.synopsys.com/detect8.ps1?$(Get-Ran
dom) | iex; detect
```

Or:

```
[Net.ServicePointManager]::SecurityProtocol = 'tls12'; $Env:DETECT_EXIT
_CODE_PASSTHRU=1; $Env:DETECT_LATEST_RELEASE_VERSION="7.0.0"; irm https
://detect.synopsys.com/detect8.ps1?$(Get-Random) | iex; detect
```

Running the Synopsys Detect .jar

Recent versions of the Synopsys Detect .jar file are available for download from the location specified in *download locations*.

To run Synopsys Detect by invoking the .jar file:

```
java -jar {path to .jar file}
```

For example:

```
curl -O https://sig-repo.synopsys.com/bds-integrations-release/com/syno
psys/integration/synopsys-detect/5.6.2/synopsys-detect-5.6.2.jar
java -jar synopsys-detect-5.6.2.jar
```

You can use the Synopsys Detect Bash script (detect8.sh) to download the Synopsys Detect .jar file:

```
export DETECT_DOWNLOAD_ONLY=1
./detect8.sh
```

Choosing the target type

Synopsys Detect will select a workflow based in part on the target type you select via the *detect.target.type* property.

When running Synopsys Detect on project source code, you'll probably want to set *detect.target.type* to *SOURCE*, or leave *detect.target.type* unset (since *SOURCE* is the default value).

When running Synopsys Detect on a Docker image, you'll probably want to set *detect.target.type* to *IMAGE*.

Common workflows

By default (detect.target.type=SOURCE), Synopsys Detect will run the following on the source directory:

1. Any applicable detectors

2. Black Duck Signature Scanner

When a Docker image is provided and property *detect.target.type* is set to IMAGE, Synopsys Detect will run the following on the image:

1. Docker Inspector
2. Black Duck Signature Scanner
3. Black Duck - Binary Analysis

Running with Black Duck

Synopsys Detect can be used with multiple Synopsys platforms and Black Duck to perform Software Composition Analysis (SCA).

Overview

Running with Black Duck and connection details are provided, Synopsys Detect executes the following by default:

- The *detector tool*, which runs the appropriate package manager-specific detector; the Maven detector for Maven projects, the Gradle detector for Gradle projects, and so forth.
- The *Black Duck Signature Scanner*, which performs a Black Duck signature scan on the project directory.
- Run *Black Duck - Binary Analysis* on given binary files.

Synopsys Detect can be configured to perform additional tasks, including the following:

- Enable any of the supported snippet matching modes in the *Black Duck Signature Scanner*.
- Enable the *Vulnerability Impact Analysis Tool* on any Java project.
- Run the Black Duck Docker Inspector on a given *Docker image*.
- Generate a *report*.
- Fail on *policy violation*.
- Run *IaC Scan* on provided targets. Note: Iac Scan capabilities require Black Duck 2022.7.0 or later.

Refer to *Black Duck Server properties*, *Black Duck Signature Scanner properties*, and *IaC Scan* for details.

Offline mode

If you do not have a Black Duck instance, or if your network is down, you can still run Synopsys Detect in offline mode. In offline mode, Synopsys Detect writes output files (.bdio files and, when Vulnerability Impact Analysis runs, .bdmu files) to subdirectories within the run directory without

attempting to upload them to Black Duck. You can find the value of the run directory in the Synopsys Detect log. You can run Synopsys Detect in offline mode using the *offline mode property*.

BDIO format

Synopsys Detect produces dependency information for Black Duck, and other Synopsys products and platforms, in Black Duck Input Output (BDIO) format files. Synopsys Detect supports generating BDIO version 2 documents.

Including and Excluding Tools, Detectors, Directories, etc.

Properties provide a variety of additional options for configuring Synopsys Detect behavior. One of the most fundamental ways to modify Synopsys Detect is by including and excluding *tools* and *detectors*.

Tools

By default, all tools are eligible to run; the set of tools that actually run depends on the properties you set. To limit the eligible tools to a given list, use:

--detect.tools={comma-separated list of tool names, all uppercase}

To exclude specific tools, use:

```
--detect.tools.excluded={comma-separated list of tool names, all upperc
ase}
```

Exclusions take precedence over inclusions.

Refer to *Tools* for the list of tool names.

Refer to *Properties* for details.

Detectors

By default, all detectors are eligible to run. The set of detectors that actually run depends on the files existing in your project directory. To limit the eligible detectors to a given list, use:

```
--detect.included.detector.types={comma-separated list of detector name
s}
```

To exclude specific detectors, use:

```
--detect.excluded.detector.types={comma-separated list of detector name
s}
```

Exclusions take precedence over inclusions.

Refer to *Detectors* for the list of detector names.

Refer to *Properties* for details.

Package Manager Exclusions

If you wish to specify package manager-specific exclusions you may do so using the following properties:

- *detect.gradle.included.configurations*
- *detect.gradle.excluded.configurations*
- *detect.gradle.included.projects*
- *detect.gradle.excluded.projects*
- *detect.lerna.included.packages*
- *detect.lerna.excluded.packages*
- *detect.maven.included.scopes*
- *detect.maven.excluded.scopes*
- *detect.maven.included.modules*
- *detect.maven.excluded.modules*
- *detect.nuget.included.modules*
- *detect.nuget.excluded.modules*

Directory Exclusions

Use *detect.excluded.directories* to exclude directories from search when looking for detectors, searching for files to binary scan when using property detect.binary.scan.file.name.patterns, and when finding paths to pass to the signature scanner as values for an '--exclude' flag.

Exclude directories by name

This property accepts explicit directory names, as well as globbing-style wildcard patterns. See *configuring property wildcards* for more info.

Examples

Value	Excluded	Not Excluded
`foo`	/projectRoot/foo	/projectRoot/foobar
`*bar`	/projectRoot/bar & /projectRoot/foobar	

Exclude directories by path

This property accepts explicit paths relative to the project's root, or you may specify glob-style patterns.

When specifying path patterns:

- Use '*' to match 0 or more directory name characters (will not cross directory boundaries).
- Use '**' to match 0 or more directory path characters (will cross directory boundaries).

Examples

Value	Excluded	Not Excluded
`foo/bar`	/projectRoot/foo/bar	/projectRoot/dir/foo/bar
`**/foo/bar`	/projectRoot/dir/foo/bar & /projectRoot/directory/foo/bar	
`/projectRoot/d*/*`	/projectRoot/dir/foo & /projectRoot/directory/bar	

Synopsys Detect uses FileSystem::getPatchMatcher and its glob syntax implementation to exclude path patterns. See *here* for more info.

Wildcards in relative path patterns

When excluding paths, if you want to use wildcards in an exclusion pattern for a relative path, there are some confusing rules.

Name-wildcards ('*'), unless appearing in a pattern that begins with path-wildcards ('**'), will only work if the pattern refers to one-level below the source path root.

Let's say you want to exclude /project/folder while scanning /project:

Value	Excluded	Not Excluded
`*older`	/project/folder	
`f*`	/project/folder	

Value	Excluded	Not Excluded
`folder/*`		/project/folder or /project/folder/dir
`**folder/*`	/project/folder/dir	/project/folder
`*older/*`		/project/folder or /project/folder/dir
`**/*older/*`	/project/folder/dir	/project/folder

Related properties:

- *detect.excluded.directories.defaults.disabled*
- *detect.excluded.directories.search.depth*
- *detect.binary.scan.file.name.patterns*

Detector search and accuracy

Detector search

Detector search is performed by the Detector tool on the source directory (`detect.source.path`). If `detect.detector.search.depth` is greater than zero, detector search is expanded to include subdirectories to the depth indicated (0 means top level only).

Detector search:

1. Finds project root directories, and
2. Determines which detector(s) should run on each project root directory that it found.

A project's root directory is the project's top-level directory viewed from the perspective of the project's package manager. There may be more than one project in your source directory.

In contrast, graph building is performed by the individual detector(s) that run on a root project. This involves building a graph that starts with the root project found by detector search. The graph includes subprojects, direct dependencies, and transitive dependencies. The graph is eventually included in Synopsys Detect's output: the BDIO. Exclusion of subsets of the graph (subprojects, configurations, etc.) is optionally done as part of graph building (controlled by properties such as `detect.{pkgmgr}.excluded.*` and `detect.{pkgmgr}.included.*`). It is important to recognize that detector search and graph building are completely separate processes controlled by different properties.

For example, directory /src/myproject might be the root directory of a Gradle project. There might also be Gradle subprojects in subdirectories underneath it (e.g. /src/myproject/subproject1), but

subprojects will be discovered during graph building. /src/myproject is the *only* directory that detector search must find.

Properties that affect detector search:

- detect.detector.search.depth
- detect.detector.search.continue

Detector search considers all the following when choosing which detector(s) to run:

1. Detector type filtering.
2. Yielding rules.
3. Nesting rules.
4. Detector cascade.

Detector type filtering

Detector type filtering honors your requests to exclude certain detector types (via properties `detect.excluded.detector.types` and `detect.included.detector.types`).

Yielding rules

Yielding rules cause some detectors to have precedence over others for a given directory. For example, if both the YARN and NPM detector types apply to a directory, only the YARN detector will apply because NPM yields to YARN.

Yielding rules cannot be disabled.

Nesting rules

While yielding rules consider which other detectors apply to the *current* directory, nesting rules consider which other detectors applied to *ancestor* directories (one or more levels up the directory path).

When detect.detector.search.depth is greater than 0, nesting rules may prevent a detector from applying on a subdirectory of the source directory (say, src/a/b/c/d) based on which detectors applied on any of its ancestor directories (src/a/b/c, src/a/b, src/a, or src).

Here are two examples of nesting rules:

1. If any GRADLE detector applied on any ancestor directory, no GRADLE detector will apply on the current directory.
2. If any XCODE detector applied on any ancestor directory, neither SWIFT detector will apply on the current directory.

Nesting rules can be disabled by setting property `detect.detector.search.continue` to true.

Detector cascade

Detector cascade is a strategy designed to produce the most accurate results possible. For a given project root directory, Detector cascade first tries the detector that would produce the most accurate results. If the first detector is unable to run (if, for example, the package manager executable it needs is not on the PATH), Detector cascade will try the next-best detector. This process continues until one of the applicable detector's extraction method succeeds or Synopsys Detect runs out of detectors that apply.

Synopsys Detect will always try the more accurate detectors first, falling back to less accurate detectors only if the more accurate detectors fail (or can't be run).

Cascade sequences are not configurable.

Detector cascade in combination with detector accuracy (described below) replace the previous (pre-Synopsys Detect 8) distinction between "build mode" and "buildless mode",

Entry points

In the *detector table*, most Detector Types have a single Entry Point. For those Detector Types, the Entry Point column can be ignored.

There are a few Detector Types for which multiple Entry Points are defined. When multiple Entry Points are defined for a Detector Type, they exist to support the relatively rare need to define different nesting rules within the same Detector Type for different scenarios. (Nesting rules can usually be, and usually are, applied at the Detector Type level.) For example: Detect should ignore XCODE project files that are nested inside an XCODE project that it has already processed. Entry Points provide the mechanism by which more nuanced nesting rules such as this are applied.

Each Entry Point has one or more Detectors. Detectors are attempted in the order listed until one applies and succeeds. If none succeed, Synopsys Detect proceeds to the next Entry Point (if there is one) for the Detector Type.

Troubleshooting detector search

For more insight into the decisions Synopsys Detect made during detector search, generate a diagnostic zip file (run with `-d`) and read the reports/search_detailed_report.txt file.

Detector execution phases

A detector has three methods:

1. The applicable method determines whether the detector applies to the current directory, based on files that it finds in the directory. For example, if a Gradle detector would look for a build.gradle file.
2. The extractable method determines whether other prerequisites are met. For example, a detector that runs a package manager executable would check to see if that executable is available.
3. The extract method discovers dependencies and returns a graph. In a few cases extraction is performed with the help of a separate Synopsys Detect component called an inspector.

Detector accuracy

Accuracy is an assessment of how complete and reliable a detector's results are. Each detector has one of two possible accuracy values: HIGH, or LOW. A detector's accuracy value is not configurable. You can find the accuracy for each detector in the *detector table*.

Detectors that run the project's package manager and discover dependencies from its output are generally assigned high accuracy because the package manager is typically a reliable source of truth about dependencies. Detectors that parse package manager-generated lockfiles also tend to be highly accurate. Detectors that parse human-editable files are generally assigned low accuracy due to challenges and limitations that are inherent in that approach.

Consider, for example, a Gradle project. Synopsys Detect could run the Gradle Native Inspector detector (which discovers dependencies by running the Gradle CLI), or the Gradle Project Inspector detector (which discovers dependencies by parsing Gradle files). If the Gradle Native Inspector succeeds, it would produce higher accuracy results than the Gradle Project Inspector detector. However, the Gradle Native Inspector may not succeed (since, for example, it must be able to find and execute a Gradle executable), and (depending on the user's preference) low accuracy might be better than nothing.

Specifying accuracy requirements

You choose the list of detector types from which you require the most accurate results using the `detect.accuracy.required` property. This property accepts a list of detector types (MAVEN, GRADLE, ...). This property defaults to ALL, which means you want Synopsys Detect to exit if any detector type applies, but only low accuracy results could be generated. When Synopsys Detect exits due to accuracy requirements not being met, it returns the FAILURE_ACCURACY_NOT_MET exit code. This default produces behavior roughly similar to the default mode (detect.detector.buildless=false) prior to Synopsys Detect 8.

To get the best results available regardless of accuracy, set this property to NONE. This value produces behavior roughly similar to buildless mode (detect.detector.buildless=true) prior to Synopsys Detect 8, except that high accuracy results will be produced where possible (buildless mode used to prevent that).

To specify that you require accurate results from some (but not all) detector types, set property `detect.accuracy.required` to the list of detector types from which you require the most accurate results.

Evaluation of accuracy

After executing detectors, an actual result accuracy is known, at which point Detect evaluates whether the detector results it was able to generate meet the user's accuracy requirements. If the user's accuracy requirements were not met, Synopsys Detect fails with the FAILURE_ACCURACY_NOT_MET exit code.

Stateless Scan LCA

Stateless Scan, or Stateless Scan Mode, is a way of running Synopsys Detect with Black Duck. This mode is designed to be as fast as possible and does not persist any data on Black Duck. Stateless Scan Mode has a unique set of restrictions, mode of configuration, and set of results. It is similar to Rapid Scan Mode, however it differs in that it supports usage of the SIGNATURE_SCAN, BINARY_SCAN, and CONTAINER_SCAN tools.

Enable this feature by adding *--detect.blackduck.scan.mode=STATELESS* to a run of Detect.

Requirements and Limitations

General Requirements

- Stateless scanning is available under Black Duck “Limited Customer Availability (LCA)”.
- Have Match as a Service (MaaS) enabled within Black Duck, a feature which will be available with the Black Duck 2022.10.0 release.
- A limited subset of Tools can be run.
  - The currently supported tools are: DETECTOR, BAZEL, SIGNATURE_SCAN, DOCKER, BINARY_SCAN, and CONTAINER_SCAN.
  - All Stateless Scans will not persist on Black Duck.
  - All other tools are disabled when running in Stateless Scan mode.
- Stateless Scan requires Black Duck policies.
  - Stateless Scan only reports components that violate policies.
  - If no policies are violated or there are no defined policies, then no components are returned.
- Stateless Scan does not support `detect.policy.check.fail.on.severities`
  - Synopsys Detect will fail with FAILURE_POLICY_VIOLATION if any component violates Black Duck polices with a CRITICAL or BLOCKER severity.
  - See the Black Duck documentation for a list of policy conditions that are supported by Stateless Scan.

- Stateless Scan does not support `detect.policy.check.fail.on.names`
- Stateless Scan cannot create a Risk or Notices report.
- Stateless Scan will not create a Project or Version on Black Duck.
- Stateless Scan when running SIGNATURE_SCAN, BINARY_SCAN, or CONTAINER_SCAN requires communication with Black Duck.
- Stateless Scan and non-persistent SIGNATURE_SCAN, BINARY_SCAN, or CONTAINER_SCAN
  - To perform a non-persistent scan in Stateless mode, SIGNATURE_SCAN, BINARY_SCAN, or CONTAINER_SCAN must be included within --detect.tools.
  - Permitted tools omitted from the detect.tools list will not be run.

Signature Scan Requirements

- Must be running Black Duck 2022.10.0 or greater using the hosted KB.

Binary and Container Scan Requirements

- Must be running Black Duck 2023.4.0 or greater using the hosted KB.
- It is necessary to have Synopsys Detect and Black Duck running in the hosted SCAaaS environment to perform these scans.
- To run binary or container Stateless Scan a Black Duck Binary Analysis (BDBA) license is required.

Invocation

- To invoke a stateless signature scan only
  - --detect.tools=SIGNATURE_SCAN --detect.blackduck.scan.mode=STATELESS
- To invoke a stateless package manager scan
  - --detect.tools=DETECTOR --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=DETECTOR --detect.blackduck.scan.mode=RAPID
  - --detect.tools=BAZEL --detect.blackduck.scan.mode=RAPID
  - --detect.tools=BAZEL --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=DOCKER --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=DOCKER --detect.blackduck.scan.mode=RAPID
  - --detect.target.type=IMAGE --detect.blackduck.scan.mode=RAPID
  - --detect.target.type=IMAGE --detect.blackduck.scan.mode=STATELESS

- To invoke a combined a stateless scan (non-exhaustive list):
  - --detect.tools=DETECTOR,SIGNATURE_SCAN --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=DETECTOR,SIGNATURE_SCAN,DOCKER --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=BAZEL,SIGNATURE_SCAN --detect.blackduck.scan.mode=STATELESS
  - --detect.tools=DETECTOR,DOCKER --detect.blackduck.scan.mode=RAPID
- To invoke a stateless binary scan
  - --detect.tools=BINARY_SCAN --detect.blackduck.scan.mode=STATELESS --detect.scaaas.scan.path=file:///foo/bar.exe
- To invoke a stateless container scan
  - --detect.tools=CONTAINER_SCAN --detect.blackduck.scan.mode=STATELESS --detect.scaaas.scan.path=file:///foo/docker-image.tar

Results

Unlike persistent scans, no data is stored on Black Duck and all scans are done transiently. These scans are primarily intended to be fast, although they can take some time as communication with Black Duck is a requirement.

The results are saved to a json file named 'name_version_BlackDuck_DeveloperMode_Result.json' in the Scan Output directory, where name and version are the project's name and version.

NOTE:

- The format of this results file is dependent on Black Duck and in the future, different versions of Black Duck may produce a different file format.

The results are also printed in the logs:

```
2021-07-20 13:25:18 EDT INFO  [main] --- Stateless Scan Result: (for mo
re detail look in the log for Stateless Scan Result Details)
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Critical and bl
ocking policy violations for
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Compo
nents: 0
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Secur
ity: 5
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Licen
```

```
se: 0
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Other policy vi
olations
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Compo
nents: 101
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Secur
ity: 0
2021-07-20 13:25:18 EDT INFO  [main] ---                        * Licen
se: 0
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Policies Violat
ed:
2021-07-20 13:25:18 EDT INFO  [main] ---                        Securit
y Vulnerabilities Great Than Or Equal to High
2021-07-20 13:25:18 EDT INFO  [main] ---                        Warn on
 Low Security Vulnerabilities
2021-07-20 13:25:18 EDT INFO  [main] ---                        Warn on
 Medium Security Vulnerabilities
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Components with
 Policy Violations:
2021-07-20 13:25:18 EDT INFO  [main] ---                        Apache
PDFBox 2.0.12 (maven:org.apache.pdfbox:pdfbox:2.0.12)
2021-07-20 13:25:18 EDT INFO  [main] ---                        Handleb
ars.js 4.0.11 (npmjs:handlebars/4.0.11)
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Components with
 Policy Violation Warnings:
2021-07-20 13:25:18 EDT INFO  [main] ---                        Acorn 5
.5.3 (npmjs:acorn/5.5.3)
```

For Synopsys Detect version 8.7.0 and later, with Black Duck 2023.1.2, Rapid Scan output now reports upgrade guidance for transitive dependencies with known vulnerabilities. The output gives information as to the direct dependency upgrade options and the transitive dependencies affected.

This output is given in the results section which appears near the end of the Synopsys Detect run and appears as follows:

```
2023-03-09 13:01:56 EST INFO  [main] --- ===== Transitive Guidance ====
=
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] ---        Transitive upgrade guid
ance:
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component c
om.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.13.2 to versi
on 2.14.2 in order to upgrade transitive components com.fasterxml.jacks
on.core:jackson-databind:2.13.2, org.yaml:snakeyaml:1.29, com.fasterxml
.jackson.core:jackson-databind:2.13.2.2, org.yaml:snakeyaml:1.30
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.apache.httpcomponents:httpclient-osgi:4.5.13 to version 4.5.14 in or
der to upgrade transitive component commons-codec:commons-codec:1.11
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.apache.pdfbox:pdfbox:2.0.25 to version 2.0.27 in order to upgrade tr
ansitive component org.apache.pdfbox:fontbox:2.0.25
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.springframework.boot:spring-boot:2.6.6 to versions (Short Term) 2.7.
9, (Long Term) 3.0.4 in order to upgrade transitive components org.spri
ngframework:spring-beans:5.3.18, org.springframework:spring-expression:
5.3.18, org.springframework:spring-aop:5.3.18, org.springframework:spri
ng-context:5.3.18
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] --- ======== Detect Status =======
=
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] --- GIT: SUCCESS
2023-03-09 13:01:56 EST INFO  [main] --- GRADLE: SUCCESS
2023-03-09 13:01:56 EST INFO  [main] --- Overall Status: SUCCESS - Dete
ct exited successfully.
2023-03-09 13:01:56 EST INFO  [main] ---
```

```
2023-03-09 13:01:56 EST INFO  [main] --- ============================
=
```

For further remediation and transitive dependency upgrade guidance, please consult the documentation provided by Black Duck under the topic: *Getting remediation guidance for components with security vulnerabilities.*

Rapid Scan

Rapid Scan or Rapid Scan Mode is a way of running Synopsys Detect with Black Duck that is designed to be as fast as possible and does not persist any data on Black Duck. Rapid Scan Mode has a unique set of restrictions, mode of configuration and set of results.

Enable this feature by adding *--detect.blackduck.scan.mode=RAPID* to a run of Detect.

Requirements and Limitations

- A limited subset of Tools can be run.
  - The currently supported tools are: DETECTOR and DOCKER.
  - All other tools are disabled when running in Rapid Scan mode.
- Rapid Scan requires Black Duck policies.
  - Rapid Scan only reports components that violate policies.
  - If no policies are violated or there are no defined policies, then no components are returned.
- Rapid Scan does not support `detect.policy.check.fail.on.severities`
  - Synopsys Detect will fail with FAILURE_POLICY_VIOLATION if any component violates Black Duck polices with a CRITICAL or BLOCKER severity.
  - See the Black Duck documentation for a list of policy conditions that are supported by Rapid Scan.
- Rapid Scan does not support `detect.policy.check.fail.on.names`
- Rapid Scan cannot create a Risk or Notices report.
- Rapid Scan will not create a Project or Version on Black Duck.

Configuration

Rapid scan policy overrides can be provided in a file named '.bd-rapid-scan.yaml' in the source directory. The file name must match exactly.

Synopsys Detect will automatically upload the config file during a rapid scan when present.

The file is a YAML file intended to be checked-in to SCM alongside other build config files.

NOTE: this file format is dependent on Black Duck and in the future, different versions of Black Duck may require a different file format.

```
version: 1.0
policy:
  overrides:
  - policyName: policyA
    components:
    - name: component1
      version: version1
    - name: component2
  - policyName: policyB
    components:
    - name: component3
      version: version3
```

Each policy override must apply to a list of specific components, on a specific version (e.g. component1 + version1) or on all versions (e.g. component2).

Results

Unlike persistent scans, no data is stored 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 named 'name_version_BlackDuck_DeveloperMode_Result.json' in the Scan Output directory, where name and version are the project's name and version.

NOTE:

- The format of this results file is dependent on Black Duck and in the future, different versions of Black Duck may produce a different file format.

The results are also printed in the logs:

```
2021-07-20 13:25:18 EDT INFO  [main] --- Rapid Scan Result: (for more d
etail look in the log for Rapid Scan Result Details)
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                Critical and bl
ocking policy violations for
```

```
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Compo
nents: 0
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Secur
ity: 5
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Licen
se: 0
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                 Other policy vi
olations
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Compo
nents: 101
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Secur
ity: 0
2021-07-20 13:25:18 EDT INFO  [main] ---                          * Licen
se: 0
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                 Policies Violat
ed:
2021-07-20 13:25:18 EDT INFO  [main] ---                          Securit
y Vulnerabilities Great Than Or Equal to High
2021-07-20 13:25:18 EDT INFO  [main] ---                          Warn on
 Low Security Vulnerabilities
2021-07-20 13:25:18 EDT INFO  [main] ---                          Warn on
 Medium Security Vulnerabilities
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                 Components with
 Policy Violations:
2021-07-20 13:25:18 EDT INFO  [main] ---                          Apache
PDFBox 2.0.12 (maven:org.apache.pdfbox:pdfbox:2.0.12)
2021-07-20 13:25:18 EDT INFO  [main] ---                          Handleb
ars.js 4.0.11 (npmjs:handlebars/4.0.11)
2021-07-20 13:25:18 EDT INFO  [main] ---
2021-07-20 13:25:18 EDT INFO  [main] ---                 Components with
 Policy Violation Warnings:
```

```
2021-07-20 13:25:18 EDT INFO  [main] ---                          Acorn 5
.5.3 (npmjs:acorn/5.5.3)
```

For Synopsys Detect version 8.7.0 and later, with Black Duck 2023.1.2, Rapid Scan output now reports upgrade guidance for transitive dependencies with known vulnerabilities. The output gives information as to the direct dependency upgrade options and the transitive dependencies affected. This output is given in the results section which appears near the end of the Synopsys Detect run and appears as follows:

```
2023-03-09 13:01:56 EST INFO  [main] --- ===== Transitive Guidance ====
=
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] ---        Transitive upgrade guid
ance:
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component c
om.fasterxml.jackson.dataformat:jackson-dataformat-yaml:2.13.2 to versi
on 2.14.2 in order to upgrade transitive components com.fasterxml.jacks
on.core:jackson-databind:2.13.2, org.yaml:snakeyaml:1.29, com.fasterxml
.jackson.core:jackson-databind:2.13.2.2, org.yaml:snakeyaml:1.30
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.apache.httpcomponents:httpclient-osgi:4.5.13 to version 4.5.14 in or
der to upgrade transitive component commons-codec:commons-codec:1.11
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.apache.pdfbox:pdfbox:2.0.25 to version 2.0.27 in order to upgrade tr
ansitive component org.apache.pdfbox:fontbox:2.0.25
2023-03-09 13:01:56 EST INFO  [main] ---            Upgrade component o
rg.springframework.boot:spring-boot:2.6.6 to versions (Short Term) 2.7.
9, (Long Term) 3.0.4 in order to upgrade transitive components org.spri
ngframework:spring-beans:5.3.18, org.springframework:spring-expression:
5.3.18, org.springframework:spring-aop:5.3.18, org.springframework:spri
ng-context:5.3.18
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] --- ======== Detect Status =======
=
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] --- GIT: SUCCESS
```

```
2023-03-09 13:01:56 EST INFO  [main] --- GRADLE: SUCCESS
2023-03-09 13:01:56 EST INFO  [main] --- Overall Status: SUCCESS - Dete
ct exited successfully.
2023-03-09 13:01:56 EST INFO  [main] ---
2023-03-09 13:01:56 EST INFO  [main] --- ==============================
=
```

For further remediation and transitive dependency upgrade guidance, please consult the documentation provided by Black Duck under the topic: *Getting remediation guidance for components with security vulnerabilities.*

Rapid Scan Compare Mode

You can configure Rapid Scan to return only the difference in policy violations between the current scan and previous Rapid Scans using the same configuration. To return only the difference in policy violations, configure detect.blackduck.rapid.compare.mode to BOM_COMPARE or BOM_COMPARE_STRICT.

Setting the compare mode to ALL evaluates all RAPID or FULL policies. BOM_COMPARE_STRICT only shows policy violations not present in an existing project version BOM. BOM_COMPARE depends on the type of policy rule modes and behaves like ALL if the policy rule is only RAPID and like BOM_COMPARE_STRICT when the policy rule is RAPID and FULL. See the Black Duck documentation for complete details.

IaC Scan

IaC ("Infrastructure as Code") Scan is a type of scanning supported by Synopsys Detect.

To configure IaC Scan, see *IaC Scan properties*.

Known Issues

Strange Output on Windows VM

Issue: When running Synopsys Detect Iac Scan on a Windows Virtual Machine, users may see in the logs for the run that the output from the IaC Scanner may look like this:

```
--- ←[4mIdentified←[0m
---
```

```
 --- ←[38;5;239mΓöé←[39mFile Type ←[38;5;239mΓöé←[39mOccurrences←[38;5;
239mΓöé←[39m
```

as supposed to this:

```
 --- Identified
 ---
 --- |File Type |Occurrences|
```

This is due to a charset encoding incompatibility between Synopsys Detect and Windows VMs.

IaC Code Location has default version

Issue: The code location generated for an IaC scan has Synopsys Detect's default version, but you know Synopsys Detect can determine your project's actual version.

Solution: Enable Detectors to run via the property detect.tools (eg. --detect.tools=DETECTOR,IAC_SCAN), and a Detector should determine your project's actual version/reflect that in the code location produced by the IaC Scan.

Running in air gap mode

To prepare to run Synopsys Detect in air gap mode, unzip the air gap archive to create the air gap directory. Do not make changes to files in the air gap directory. Invoke the Synopsys Detect .jar file from its original unzipped location at the top level of the air gap directory. For more information on invoking the .jar file, refer to *Running the Synopsys Detect .jar*.

Adding the Black Duck Signature Scanner to your air gap archive

To create an air gap archive that includes the Black Duck Signature Scanner, follow these steps:

1. Unzip the Synopsys Detect air gap archive to create the Synopsys Detect air gap directory.
2. Download the appropriate Black Duck Signature Scanner zip file from your Black Duck instance (System > Tools > Legacy Downloads > Signature Scanner), and unzip it. This will create a directory with a name like scan.cli-x.y.z.
3. Move that scan.cli-x.y.z directory to the top level of the Synopsys Detect air gap directory.
4. Zip the enhanced Synopsys Detect air gap directory to create your enhanced air gap archive.

When you later run Synopsys Detect from the directory created by unzipping your enhanced air gap archive, set property detect.blackduck.signature.scanner.local.path to the path to the scan.cli-x.y.z directory in your enhanced air gap archive directory.

Status File

Synopsys Detect creates an output status file in the run folder with the name "status.json" which contains a summary of the Detect run in a machine readable format.

The file includes status codes, issues encountered and results produced. As additional processes consume this file, additional information will be added. The format is intended to evolve over time.

- As Detect shuts down, by default, it performs cleanup operations which include deleting the status file. You can disable clean up by setting `--detect.cleanup=false`.

Body

```
{
"formatVersion": The version of the status file format. Will change as
new features are introduced.
"detectVersion": The version of [solution_name] that created the status
 file.
"projectName": The project name.
"projectVersion": The project version.
"detectors": [ List of Detectors, see details below. ]
"status": [ List of Status, see details below. ]
"issues": [ List of Issues, see details below. ]
"overallStatus: [ List the overall exit status and detailed message on
exit of Detect. ]
"results": [ List of Results, see details below. ]
"unrecognizedPaths": [ List of Unrecognized Paths, see details below. ]
"codeLocations": [ List of code locations produced, see details below.
]
"propertyValues": { An object representing all provided properties, see
 details below. }
"operations": [ List of performed operations, see details below. ]
}
```

Detector

```
{
"folder": The folder the detector applied to.
"detectorType": The normalized detector type such as "GIT".
"detectorName": A shorthand name of the detector such as "Git Cli".
"discoverable":  A boolean indicating whether or not the detector was a
ble to discover project information.
"extracted": A boolean indicating whether or not the detector was able
to extract dependencies.
"status": An enum indicating whether the detector was successful, faile
d, or deferred to another detector.
"statusCode": A code specifying the nature of the detector's failure, o
r PASSED if the detector was successful. See below for a complete list
of possible status codes.
"statusReason": A human readable description of the status code.
"relevantFiles": [ A list of files relevant to the detector. ]
"discoveryReason": A human readable description of the discovery result
.
"extractedReason": A human readable description of the extraction resul
t.
"projectName": The project name this detectable found.
"projectVersion": The project version this detectable found.
"codeLocationCount": The number of code locations this detector produce
d.
"explanations": [ A human readable list of strings describing why this
detector ran such as "Found file:
<path>". ]
}
```

#Detector Status Codes

Status Code	Description
ATTEMPTED	Detector attempted to run but did not succeed.
CARGO_LOCKFILE_NOT_FOUND	A Cargo.toml was located in the target project, but the Cargo.lock file was NOT located.
CARTFILE_RESOLVED_FILE_NOT_FOUND	A Cartfile was located in the target project, but the Cartfile.resolved file was NOT located.
EXCEPTION	An exception occurred.
EXCLUDED	Detector type was excluded.
EXECUTABLES_NOT_FOUND	The necessary executables were not found.
EXECUTABLE_FAILED	During extraction, one or more executables did not execute successfully.
EXECUTABLE_NOT_FOUND	The necessary executable was not found.
EXTRACTION_FAILED	During extraction, one or more exceptions were encountered.
FAILED	Detector failed.
FILES_NOT_FOUND	Necessary files were not found within the target project.
FILE_NOT_FOUND	A file was not found within the target project.
FORCED_NESTED_PASSED	Forced to pass because nested forced by user.
GO_PKG_LOCKFILE_NOT_FOUND	A Gopkg.toml was located in the target project, but the Gopkg.lock file was NOT located.
INSPECTOR_NOT_FOUND	The necessary inspector was not found
MAX_DEPTH_EXCEEDED	Max depth was exceeded.
NOT_NESTABLE	Not nestable and a detector already applied in parent directory.
NOT_NESTABLE_BENEATH	Nestable but another detector prevented nesting.
NPM_NODE_MODULES_NOT_FOUND	A package.json was located in the target project, but the node_modules folder was NOT located.
PASSED	Detector passed.
PIPFILE_LOCK_NOT_FOUND	A Pipfile was located in the target project, but a Pipfile.lock was NOT located.

Status Code	Description
POETRY_LOCKFILE_NOT_FOUND	A pyproject.toml was located in the target project, but the Poetry.lock file was NOT located.
PROPERTY_INSUFFICIENT	The properties are insufficient to run.
PUBSPEC_LOCK_NOT_FOUND	A pubspec.yaml file was found, but a pubspec.lock file was NOT found.
SBT_PLUGIN_MISSING	A dependency graph plugin must be installed for the SBT detector to run.
SECTION_NOT_FOUND	A necessary section was not found within a file within the target project.
UNKNOWN_DETECTOR_RESULT	There was an unknown result.
WRONG_OPERATING_SYSTEM_RESULT	Cannot run on the used operating system.
YIELDED	Yielded to other detectors.

Status

```
{
"key": The normalized key this status element describes such as "GIT".
"status": "SUCCESS" or "FAILURE"
}
```

Issues

```
{
"type": A key describing the type of issue, currently "EXCEPTION", "DEP
RECATION" or "DETECTOR".
"title": A string describing the issue.
"messages": A list of a strings describing the details of the issue.
}
```

Results

A result is a URL, file path to output, or messages produced by the Synopsys Detect run: a Black Duck Bill Of Materials, Risk Report, Notices Report, Air Gap zip, or Rapid Scan results.

```
{
"location": The path to the result.
"message": A string describing the result.
"sub_messages": A list of strings providing more detail about the resul
t.
}
```

Unrecognized Paths

For those detectors that support it (currently, only CLANG), a list of file paths to dependencies that (a) were not recognized by the package manager, and (b) reside outside the source directory.

```
{
"<Detector type>": [ A list of file paths to unrecognized dependencies
]
}
```

Code Locations

```
{
"codeLocationName": The name of a code location produced by this run of
 [solution_name].
}
```

Property Values

A map of every property key to it's string value that Detect found. These are only properties to which Detect has a known key, so pass-through properties like Docker and dynamic properties like custom fields are not included. Passwords and other sensitive fields are masked.

```
  "propertyValues": {
    "key": "value",
    "boolean-key": "true"
  }
```

Operations

A list of information regarding internal execution of Detect to describe when portions of Detect run and what their status is. This information is intended to be used when Detect fails and the reason(s) for a Detect failure.

```
  "operations": {
    "startTimestamp": A formatted UTC timestamp when the execution star
ted.
    "endTimestamp": A formatted UTC timestamp when the execution ended.
    "descriptionKey": A string that describes what is being executed.
    "status": "SUCCESS" or "FAILURE"
  }
```

Running behind a proxy

When running behind a proxy:

1. The one-liner cannot be used to download the scripts, they are not proxy aware. The scripts must already be downloaded.
2. The script (detect8.sh or detect8.ps1) requires proxy details to do a version check on, and/or download the Synopsys Detect .jar file.
3. Synopsys Detect; the code in the .jar file, requires proxy details to download inspectors and connect to Black Duck.

Providing proxy details to Synopsys Detect

Synopsys Detect looks for proxy details in the properties whose names start with `blackduck.proxy`, including:

- `blackduck.proxy.host` (proxy hostname)
- `blackduck.proxy.port` (proxy port number)
- `blackduck.proxy.username` (proxy username)
- `blackduck.proxy.password` (proxy password)

When setting the blackduck.proxy.host (proxy hostname) property, the schema/protocol is not accepted.

For example:

```
Correct: `--blackduck.proxy.host=<Proxy_IP/URL>`
Incorrect: `--blackduck.proxy.host=<https ://(IP/Server_URL)>`
```

Refer to *properties* for more information.

Providing proxy details to detect8.sh

The curl commands executed by detect8.sh to do a version check on, and/or download the Synopsys Detect .jar file, require additional command line options when run behind a proxy. For more information on curl options, refer to the *curl documentation*.

To provide additional curl command line options for detect8.sh to use when it executes curl, set the environment variable DETECT_CURL_OPTS before running detect8.sh. For example:

```
export DETECT_CURL_OPTS=--proxy http://myproxy:3128
./[bash_script_name]
```

When using detect8.sh to execute Synopsys Detect you must set proxy properties for Synopsys Detect as previously described.

Providing proxy details to detect8.ps1

detect8.ps1 derives proxy details from environment variables whose names match the Synopsys Detect proxy property names. Configuring detect8.ps1 for your proxy involves setting those environment variables before running detect8.ps1. Note that typically, the PowerShell script is run

from a Command window, using "powershell script.ps1" so these should be run in that Command window. For example:

```
${r"set BLACKDUCK_PROXY_HOST"}=$ProxyHost
${r"set BLACKDUCK_PROXY_PORT"}=$ProxyPort
${r"set BLACKDUCK_PROXY_PASSWORD"}=$ProxyUsername
${r"set BLACKDUCK_PROXY_USERNAME"}=$ProxyPassword
powershell "Import-Module FULL_PATH_TO_DOWNLOADED_SCRIPT/detect8.ps1; d
etect"
```

For additional information on these properties, including alternate key formats, see the *Shell script configuration reference*.

When using detect8.ps1 to execute Synopsys Detect, Synopsys Detect also receives the proxy details from these environment variables, so no additional configuration is required for Synopsys Detect.

Concurrent execution

Concurrent execution of Synopsys Detect by the same user can result in collisions as the Synopsys Detect script, the Synopsys Detect .jar, the Synopsys Detect inspectors, and the Black Duck Signature Scanner are each downloaded to the same default location during execution. There are also potential race conditions that can occur when multiple concurrent runs of Synopsys Detect create or update the same Black Duck project/version or codelocation.

Concurrent execution of Synopsys Detect runs that include Docker image inspection involves additional challenges. For that scenario, we recommend engaging Synopsys Software Integrity Group Client Services for a solution tailored to your environment. The rest of this page addresses scenarios that do not involved inspecting Docker images.

The recommended way for a single user to execute multiple Synopsys Detect runs concurrently and avoid the collisions mentioned above is to:

1. Run Synopsys Detect using the *air gap* capability. This avoids downloading the Synopsys Detect script, .jar, or inspectors during execution.
2. Manually download and install the Black Duck Signature Scanner, and point Synopsys Detect to it. This avoids downloading the Black Duck Signature Scanner during execution.
3. Ensure that concurrent runs do not attempt to create or update the same Black Duck project/version, or the same codelocation.

To accomplish the first two:

1. Log into Black Duck, and under Tools > Legacy Downloads, download and unzip the Black Duck Signature Scanner.

2. Download the Synopsys Detect "no docker" air gap zip from the location specified in *download locations*, and unzip it. More details on using air gap mode can be found on the *air gap page*.
3. Run Synopsys Detect as shown in this example:

```
java -jar {airgap dir}/synopsys-detect-{version}.jar --detect.blackduck
.signature.scanner.local.path={scan.cli-yourBlackDuckVersion dir}
```

Running Synopsys Detect from within a Docker container

Synopsys Detect publishes Docker images which can be used to run Synopsys Detect from within a Docker container.

To Use

To run a container built from a Synopsys Detect image, use the Docker CLI's `docker run` command.

- Use the -it options to view logs during the container run.
- Use the -v option to create a bind mount that will link a provided path to project source on your host to the /source directory within the container. Do this in place of providing the --detect.source.path property, as you would when running Synopsys Detect via the script or jar.
- You may also use the -v option to create a bind mount that will link a provided path to an output directory on your host to the /output directory within the container. Do this in place of providing the --detect.output.path property, as you would when running Synopsys Detect via the script or jar.
- Use the --rm option to clean up the container once it exits.
- Provide Synopsys Detect property values as you would when running via the Synopsys Detect script or the Synopsys Detect jar, at the end of the `docker run` command.

Find available images *here*.

Find the source for them (Dockerfiles) *here*.

The format of image names is: `blackducksoftware/detect:[detect_version]-[package_manager]-[package_manager_version]`

- If you want an image with the latest supported release for a major version of Synopsys Detect, and the latest supported version of a package manager, such images are named in the following format: `blackducksoftware/detect:[detect_major_version]-[package_manager]`

Synopsys Detect Basic Images

If you wish to build your own custom Synopsys Detect image, to run Synopsys Detect in buildless mode, or to run non-detector tools such as the Signature Scanner or Binary Scanner, there also exist "simple" Synopsys Detect images. These images contain no package manager files or executables.

The format of "simple" image names is: `blackducksoftware/detect:[detect_version]`

- If you want an image with the latest supported release for a major version of Synopsys Detect, such images are named in the following format: `blackducksoftware/detect:[detect_major_version]`

Synopsys Detect Buildless Images

There also exist "buildless" Synopsys Detect images. These images automatically pass the argument --detect.accuracy.required=NONE when running to make Synopsys Detect as resilient as possible (it will evaluate all applicable detectors, regardless of their accuracy, in order to get results).

The format of "buildless" image names is: `blackducksoftware/detect:[detect_version]-buildless`

- If you want a buildless image with the latest supported release for a major version of Synopsys Detect, such images are named in the following format: `blackducksoftware/detect:[detect_major_version]-buildless`

Synopsys Detect IaC Images

If you wish to perform an IaC Scan via Synopsys Detect in a Docker container, there exist "iac" Synopsys Detect images. The scanner that Synopsys Detect uses to perform IaC scans is not supported in other Synopsys Detect images.

The format of "iac" image names is: `blackducksoftware/detect:[detect_version]-iac`

- If you want an iac image with the latest supported release for a major version of Synopsys Detect, such images are named in the following format: `blackducksoftware/detect:[detect_major_version]-iac`

Examples

```
docker run -it --rm -v [/path/to/source]:/source -v [/path/to/outputDir]:/
output blackducksoftware/detect:[detect_image_tag] [detect_arguments]
```

```
docker run -it --rm -v /Home/my/gradle/project:/source -v /Home/where/I/
want/detect/output/files:/output blackducksoftware/detect:7.0.0 --
blackduck.url=https://my.blackduck.url --blackduck.api.token=MyT0kEn
```

```
docker run -it --rm -v /Home/my/maven/project:/source -v /Home/where/I/
want/detect/output/files:/output blackducksoftware/detect:7 --
blackduck.url=https://my.blackduck.url --blackduck.api.token=MyT0kEn
```

```
docker run -it --rm -v /Home/my/project:/source -v /Home/where/I/want/
detect/output/files:/output blackducksoftware/detect:7.0.0 --
blackduck.url=https://my.blackduck.url --blackduck.api.token=MyT0kEn --
detect.detector.buildless=true
```

```
docker run -it --rm -v /Home/my/project:/source -v /Home/where/I/want/
detect/output/files:/output blackducksoftware/detect:6.9.1 --
blackduck.url=https://my.blackduck.url --blackduck.api.token=MyT0kEn --
detect.tools=SIGNATURE_SCAN,BINARY_SCAN
```

9

Synopsys Detect Components

This page introduces the components in Synopsys Detect that are used to examine your code so that the output from one or more of these can be analyzed.

The components comprise the following:

Tools

Each Synopsys Detect run consists of running any applicable Synopsys Detect tools.

Detectors

Synopsys Detect uses detectors to find and extract dependencies from all supported package managers.

Inspectors

An inspector is typically a plugin that a Synopsys Detect detector uses to access the internal resources of a package manager through its API.

Tools

Each Synopsys Detect run consists of running any applicable Synopsys Detect tools.

The available Synopsys Detect tools in order of execution, with the corresponding *detect tools property* value specified in parentheses are:

- *Docker Inspector* (--detect.tools=DOCKER)
- *Bazel* (--detect.tools=BAZEL)
- *Detector* (--detect.tools=DETECTOR)
- *Black Duck Signature Scanner* (--detect.tools=SIGNATURE_SCAN)
- *Black Duck - Binary Analysis* (--detect.tools=BINARY_SCAN)
- *Vulnerability Impact Analysis Tool* (--detect.tools=IMPACT_ANALYSIS)
- *IaC Scanner* (--detect.tools=IAC_SCAN)

The detector tool runs any applicable *detectors*.

Detectors

The Synopsys Detect Detector tool runs one or more detectors to find and extract dependencies from all supported package managers.

Each package manager ecosystem is assigned a detector type. Each detector type may have multiple methods (detectors) used to extract dependencies.

Which detector(s) will run against your project is determined by the *detector search* process.

Detector Types, Entry Points, and Detectors

The following table contains details for each Detector type, entry point, and detector. Details on these terms is available on the *detector search* page.

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
BITBAKE						
	Bitbake CLI					
		Bitbake CLI	various	YOCTO	Properties: Package names. File: build env script. Executable: bash	HIGH
CARGO						
	Cargo Lock					
		Cargo Lock	Rust	crates	Files: Cargo.lock, Cargo.toml	HIGH
CARTHAGE						
	Carthage Lock					
		Carthage Lock	various	GitHub	Files: Cartfile, Cartfile.resolved	HIGH
CLANG						

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
	Clang CLI					
		Clang CLI	C or C++	Derived from the Linux distribution.	File: compile_commands.json. Executable: Linux package manager.	HIGH
COCOAPODS						
	Pod Lock					
		Pod Lock	Objective C	COCOAPODS and NPMJS	Files: Podfile.lock	HIGH
CONAN						
	Conan Lock					
		Conan Lock	C/C++	conan	Files: conan.lock.	HIGH
		Conan CLI	C/C++	conan	Files: conanfile.txt or conanfile.py. Executable: conan.	HIGH
	Conan CLI					
		Conan CLI	C/C++	conan	Files: conanfile.txt or conanfile.py. Executable: conan.	HIGH
CONDA						
	Conda CLI					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		Conda CLI	Python	Anaconda	File: environment.yml. Executable: conda.	HIGH
CPAN						
	Cpan CLI					
		Cpan CLI	Perl	CPAN	File: Makefile.PL. Executable: cpan.	HIGH
CRAN						
	Packrat Lock					
		Packrat Lock	R	CRAN	File: packrat.lock.	HIGH
DART						
	Dart CLI					
		Dart CLI	Dart	Dart	Files: pubspec.yaml, pubspec.lock. Executable: dart, flutter	HIGH
		Dart PubSpec Lock	Dart	Dart	Files: pubspec.yaml, pubspec.lock.	HIGH
	Dart PubSpec Lock					
		Dart PubSpec Lock	Dart	Dart	Files: pubspec.yaml, pubspec.lock.	HIGH
GIT						

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
	Git					
		Git	various	N/A	Directory: .git. (Executable: git).	HIGH
		Git Parse	various	N/A	Files: .git/config, .git/HEAD, .git/ORIG_HEAD.	HIGH
	Git Parse					
		Git Parse	various	N/A	Files: .git/config, .git/HEAD, .git/ORIG_HEAD.	HIGH
GO_DEP						
	GoDep Lock					
		GoDep Lock	Golang	GitHub	File: Gopkg.lock.	HIGH
GO_GRADLE						
	GoGradle Lock					
		GoGradle Lock	Golang	GitHub	File: gogradle.lock.	HIGH
GO_MOD						
	GoMod CLI					
		GoMod CLI	Golang	Go Modules	File: go.mod. Executable: go.	HIGH
GO_VENDOR						
	Go Vendor					
		Go Vendor	Golang	GitHub	File: vendor/vendor.json.	HIGH

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
GO_VNDR						
	GoVndr CLI					
		GoVndr CLI	Golang	GitHub	File: vendor.conf.	HIGH
GRADLE						
	Gradle Native Inspector					
		Gradle Native Inspector	various	Maven Central	File: build.gradle or build.gradle.kts. Executable: gradlew or gradle.	HIGH
		Gradle Project Inspector	various	Maven Central	File: build.gradle	LOW
HEX						
	Rebar CLI					
		Rebar CLI	Erlang	Hex	File: rebar.config. Executable: rebar3.	HIGH
IVY						
	Ivy Build Parse					
		Ivy Build Parse	various	Maven Central	File: ivy.xml, build.xml.	LOW
LERNA						
	Lerna CLI					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		Lerna CLI	Node JS	npmjs	File: lerna.json, package.jso n, Executable: Lerna, and one of the following: package-lock.json, npm-shrinkwrap.j son, yarn.lock.	HIGH
MAVEN						
	Maven CLI					
		Maven CLI	various	Maven Central	File: pom.xml. Executable: mvnw or mvn.	HIGH
		Maven Project Inspector	various	Maven Central	File: pom.xml.	LOW
	Maven Wrapper CLI					
		Maven Wrapper CLI	various	Maven Central	File: pom.groovy. Executable: mvnw or mvn.	HIGH
		Maven Project Inspector	various	Maven Central	File: pom.xml.	LOW
NPM						
	NPM Shrinkwrap					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		NPM Shrinkwrap	Node JS	npmjs	File: npm-shrinkwrap.json. Optionally for better results: package.json also.	HIGH
		NPM Package Lock	Node JS	npmjs	File: package-lock.json. Optionally for better results: package.json also.	HIGH
		NPM CLI	Node JS	npmjs	Files: node_modules, package.json. Executable: npm.	HIGH
		NPM Package Json Parse	Node JS	npmjs	File: package.json.	LOW
	NPM Package Lock					
		NPM Package Lock	Node JS	npmjs	File: package-lock.json. Optionally for better results: package.json also.	HIGH

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		NPM CLI	Node JS	npmjs	Files: node_modules, package.json. Executable: npm.	HIGH
		NPM Package Json Parse	Node JS	npmjs	File: package.json.	LOW
	NPM CLI					
		NPM CLI	Node JS	npmjs	Files: node_modules, package.json. Executable: npm.	HIGH
		NPM Package Json Parse	Node JS	npmjs	File: package.json.	LOW
	NPM Package Json Parse					
		NPM Package Json Parse	Node JS	npmjs	File: package.json.	LOW
NUGET						
	NuGet Solution Native Inspector					
		NuGet Solution Native Inspector	C#	NuGet.org	File: a solution file with a .sln extension.	HIGH

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		NuGet Project Inspector	C#	NuGet.org	File: a project file with one of the following extensions: .csproj, .sln	LOW
	NuGet Project Native Inspector					
		NuGet Project Native Inspector	C#	NuGet.org	File: a project file with one of the following extensions: .csproj, .fsproj, .vbproj, .asaproj, .dcproj, .shproj, .ccproj, .sfproj, .njsproj, .vcxproj, .vcproj, .xproj, .pyproj, .hiveproj, .pigproj, .jsproj, .usqlproj, .deployproj, .msbuildproj, .sqlproj, .dbproj, .rproj	HIGH
		NuGet Project Inspector	C#	NuGet.org	File: a project file with one of the following extensions: .csproj, .sln	LOW
PACKAGIST						
	Composer Lock					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		Composer Lock	PHP	Packagist.org	Files: composer.lock, composer.json.	HIGH
PEAR						
	Pear CLI					
		Pear CLI	PHP	Pear	Files: package.xml. Executable: pear.	HIGH
PIP						
	Pipenv Lock					
		Pipenv Lock	Python	PyPi	Files: Pipfile or Pipfile.lock. Executables: python or python3, and pipenv.	HIGH
		PIP Native Inspector	Python	Pypi	A setup.py file, or one or more requirements.txt files. Executables: python and pip, or python3 and pip3.	HIGH
		Pipfile Lock	Python	PyPi	Files: Pipfile, Pipfile.lock	HIGH
	PIP Native Inspector					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		PIP Native Inspector	Python	Pypi	A setup.py file, or one or more requirement s.txt files. Executables: python and pip, or python3 and pip3.	HIGH
		Pipfile Lock	Python	PyPi	Files: Pipfile, Pipfile.lock	HIGH
	Pipfile Lock					
		Pipfile Lock	Python	PyPi	Files: Pipfile, Pipfile.lock	HIGH
PNPM						
	Pnpm Lock					
		Pnpm Lock	Node JS	npmjs	Files: pnpm-lock.yaml and package.jso n.	HIGH
POETRY						
	Poetry Lock					
		Poetry Lock	Python	pypi	Files: Poetry.lock, pyproject.to ml	HIGH
RUBYGEMS						
	Gemfile Lock					
		Gemfile Lock	Ruby	RubyGems	File: Gemfile.lock.	HIGH
		Gemspec Parse	Ruby	RubyGems	File: A gemspec file (with .gemsp ec extension).	LOW

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
	Gemspec Parse					
		Gemspec Parse	Ruby	RubyGems	File: A gemspec file (with .gemspec extension).	LOW
SBT						
	Sbt Native Inspector					
		Sbt Native Inspector	Scala	Maven Central	File: build.sbt. Plugin: Dependency Graph	HIGH
SWIFT						
	Swift Lock					
		Swift Lock	Swift	Swift.org	File: Package.swift, Package.resolved	HIGH
		Swift CLI	Swift	Swift.org	File: Package.swift. Executables: swift.	HIGH
	Swift CLI					
		Swift CLI	Swift	Swift.org	File: Package.swift. Executables: swift.	HIGH
XCODE						
	Xcode Workspace Lock					

Detector Typ	Entry Point	Detector	Language	Forge	Requiremen	Accuracy
		Xcode Workspace Lock	Swift	GITHUB	Directory: *.xcworkspace	HIGH
		Xcode Project Lock	Swift	GITHUB	Directory: *.xcodeproj, Files: Package.resolved	HIGH
	Xcode Project Lock					
		Xcode Project Lock	Swift	GITHUB	Directory: *.xcodeproj, Files: Package.resolved	HIGH
YARN						
	Yarn Lock					
		Yarn Lock	Node JS	npmjs	Files: yarn.lock and package.json.	HIGH

Inspectors

An inspector is typically a plugin that a Synopsys Detect detector uses to access the internal resources of a package manager through its API.

There are currently three inspectors that Synopsys Detect might download and one inspector it has internally.

If Synopsys Detect decides that your package manager needs an external inspector you must either be online or have the applicable air gap files.

Gradle inspector

For Gradle a common integrations library is added as a dependency to a temporary Gradle script file that is executed by the Synopsys Detect Gradle detector.

If you are online the Synopsys Artifactory instance is added as a Maven repository and the library is downloaded by Gradle. If you are offline the air gap library jar files are added as classpath file dependencies.

In both cases, the Synopsys Detect Gradle detector executes the custom Gradle script mentioned above, which invokes a custom Gradle task.

The source code for the library is located at *GitHub*.

Docker Inspector

The Docker Inspector is available as a Java jar file or shell script for Linux or Mac.

If you are online the Synopsys Artifactory instance is used to download the Docker Inspector jar file.

If you are offline the Docker Inspector jar file and required Docker image tar files are sourced from the provided path to the Docker Inspector air gap files. Synopsys Detect loads the Docker images (container-based services that Docker Inspector depends on) from the provided image tar files so they are available to the Docker Inspector.

In either case the Docker Inspector jar is run by default which uses your Docker engine to start and stop the container-based services.

The source code for Docker Inspector is located at *GitHub*.

NuGet Inspector

The NuGet inspector is available as an independent executable for Windows, Linux and Mac.

If you are online the Synopsys Artifactory instance is used to download the applicable NuGet inspector which is then unzipped and the runtime files are located. If you are offline the air gap inspector runtime files are located at the provided path.

In either case the located executable is run which communicates with NuGet and the dotnet build system.

The source code for the NuGet inspector is located at *GitHub*.

Python Inspector

While Python has an inspector this inspector is not downloaded from an external source and is contained in the Synopsys Detect source code.

The Python Inspector is a Python script that Synopsys Detect executes using Python.

This script uses pip's internal methods to extract dependencies.

10

Package Manager Information

This section contains more detailed information about how many of the supported package managers operate.

Refer to the Package Manager menu on the left for the list of package managers and their respective details.

To view the package manager versions that Synopsys Detect supports, please refer to the *Release Compatibility Matrix.*

Bazel support

Related properties

Detector properties

Overview

Synopsys Detect provides limited support for Bazel projects.

Synopsys Detect discovers dependencies specified in *maven_jar*, *maven_install*, *haskell_cabal_library* workspace rules. It also discovers library dependencies that have a github released artifact location (URL) specified in an *http_archive* workspace rule (see below for details).

The Synopsys Detect Bazel tool attempts to run on your project if you provide a Bazel build target using the Bazel target property.

The Bazel tool also requires a bazel executable on $PATH.

By default, Synopsys Detect determines the set of supported workspace rules that your project uses by parsing the WORKSPACE file, and executes Bazel commands to discover dependencies for each supported workspace rule it finds. Alternatively, you can directly control the set of workspace rules Synopsys Detect uses by setting the Bazel workspace rule property. Refer to *Properties* for details.

Processing for the *maven_install* workspace rule

The Bazel tool runs a bazel cquery on the given target to produce output from which it can parse artifact details such as group, artifact, and version for dependencies.

Synopsys Detect's Bazel detector uses commands very similar to the following to discover *maven_install* dependencies.

```
$ bazel cquery --noimplicit_deps 'kind(j.*import, deps(//tests/integrat
ion:ArtifactExclusionsTest))' --output build 2>&1 | grep maven_coordina
tes
tags = ["maven_coordinates=com.google.guava:guava:27.0-jre"],
tags = ["maven_coordinates=org.hamcrest:hamcrest:2.1"],
tags = ["maven_coordinates=org.hamcrest:hamcrest-core:2.1"],
tags = ["maven_coordinates=com.google.guava:listenablefuture:9999.0-emp
ty-to-avoid-conflict-with-guava"],
tags = ["maven_coordinates=org.checkerframework:checker-qual:2.5.2"],
tags = ["maven_coordinates=com.google.guava:failureaccess:1.0"],
tags = ["maven_coordinates=com.google.errorprone:error_prone_annotation
s:2.2.0"],
tags = ["maven_coordinates=com.google.code.findbugs:jsr305:3.0.2"],
```

Then, it parses the group/artifact/version details from the values of the maven_coordinates tags.

Processing for the *maven_jar* workspace rule

The Bazel tool runs a bazel query on the given target to get a list of jar dependencies. On each jar dependency, the Bazel tool runs another bazel query to get its artifact details: group, artifact, and version.

The following is an example using the equivalent commands that Synopsys Detect runs, but from the command line, showing how Synopsys Detect's Bazel detector currently identifies components. First, it gets a list of dependencies:

```
$ bazel cquery 'filter("@.*:jar", deps(//:ProjectRunner))'
INFO: Invocation ID: dfe8718d-b4db-4bd9-b9b9-57842cca3fb4
@org_apache_commons_commons_io//jar:jar
@com_google_guava_guava//jar:jar
Loading: 0 packages loaded
```

Then, it gets details for each dependency. It prepends //external: to the dependency name for this command.

```
$ bazel query 'kind(maven_jar, //external:org_apache_commons_commons_io
)' --output xml
```

```
INFO: Invocation ID: 0a320967-b2a8-4b36-ab47-e183bc4d4781
<?xml version="1.1" encoding="UTF-8" standalone="no"?>
<query version="2">
    <rule class="maven_jar" location="/root/home/steve/examples/java-tu
torial/WORKSPACE:6:1" name="//external:org_apache_commons_commons_io">
        <string name="name" value="org_apache_commons_commons_io"/>
        <string name="artifact" value="org.apache.commons:commons-io:1.
3.2"/>
    </rule>
</query>
Loading: 0 packages loaded
```

Finally, it parses the group/artifact/version details from the value of the string element using the name of artifact.

Processing for the *haskell_cabal_library* workspace rule

Requires Bazel 2.1.0 or later.

Synopsys Detect's Bazel detector runs a bazel cquery on the given target to produce output from which it can extract artifact project and version for dependencies.

The Bazel detector uses a command very similar to the following to discover *haskell_cabal_library* dependencies.

```
$ bazel cquery --noimplicit_deps 'kind(haskell_cabal_library, deps(//ca
t_hs/lib/args:args))' --output jsonproto
{
"results": [{
"target": {
"type": "RULE",
"rule": {
...
"attribute": [{
...
}, {
"name": "name",
"type": "STRING",
```

```
"stringValue": "hspec",
"explicitlySpecified": true,
"nodep": false
}, {
"name": "version",
"type": "STRING",
"stringValue": "2.7.1",
"explicitlySpecified": true,
"nodep": false
}, {
...
```

It then uses Gson to parse the JSON output into a parse tree, and extracts the name and version from the corresponding rule attributes.

Processing for the *http_archive* workspace rule

Synopsys Detect discovers library dependencies that have a github released artifact location (URL) specified in an *http_archive* workspace rule.

One of the URLs provided in the *http_archve* rule must match one of the following two forms:

- https://github.com/{organization}/{repo}/archive/{version}.{extension}
- https://github.com/{organization}/{repo}/releases/download/{version}/{filename}.{extension}

Supported extensions: .tar.gz, .zip, .gz

The Bazel tool runs a bazel query on the given target to get a list of library dependencies. On each dependency in the output, the Bazel tool runs another bazel query to get its artifact details; specifically the github released artifact locations (URLs) within http_archive, go_repository, and git_repository rules.

The following is an example using the equivalent commands that Synopsys Detect runs, but from the command line, showing how Synopsys Detect's Bazel detector currently identifies components. First, it gets a list of dependencies:

```
# bazel query 'kind(.*library, deps(//:bd_bazel))'
Starting local Bazel server and connecting to it...
@bazel_tools//third_party/def_parser:def_parser_lib
@bazel_tools//tools/cpp:malloc
@com_github_gflags_gflags//:gflags
```

```
@glog//:glog
Loading: 11 packages loaded
```

Then, it gets details for each http_archive dependency. It prepends //external: to the dependency name and runs a command similar to this:

```
# bazel query 'kind(.*, //external:com_github_gflags_gflags)' --output
xml
<?xml version="1.1" encoding="UTF-8" standalone="no"?>
<query version="2">
    <rule class="http_archive" location="/home/steve/src/bd_bazel/WORKS
PACE:4:13" name="//external:com_github_gflags_gflags">
        <string name="name" value="com_github_gflags_gflags"/>
        <list name="urls">
            <string value="https://github.com/gflags/gflags/archive/v2.
2.2.tar.gz"/>
        </list>
        <string name="sha256" value="34af2f15cf7367513b352bdcd2493ab14c
e43692d2dcd9dfc499492966c64dcf"/>
        <string name="strip_prefix" value="gflags-2.2.2"/>
    </rule>
</query>
Loading: 0 packages loaded
```

Finally, for each http_archive, go_repository, and git_repository rule in the output, Synopsys Detect parses the github dependency name (organization/repo) and version from each github released artifact location (in the url field or urls list, whichever is present).

Examples

mvn_install rule example

The following example will (if you add your Black Duck connection details to the Synopsys Detect command line) run the Bazel tool on the *//tests/integration:ArtifactExclusionsTest* target in the rules_jvm_external project and discover dependencies defined with the maven_install repository rule:

```
git clone https://github.com/bazelbuild/rules_jvm_external
cd rules_jvm_external/
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.baze
l.target='//tests/integration:ArtifactExclusionsTest'
```

haskell_cabal_library rule example

The following example will (if you add your Black Duck connection details to the Synopsys Detect command line) run the Bazel tool on the *//cat_hs/lib/args:args* target in the rules_haskell/examples project and discover dependencies defined with the haskell_cabal_library repository rule:

```
git clone https://github.com/tweag/rules_haskell.git
cd rules_haskell/examples
bash <(curl -s -L https://detect.synopsys.com/detect8.sh) --detect.baze
l.target='//cat_hs/lib/args:args'
```

BitBake support

Related properties

Detector properties

Requirements

The BitBake detector will run if it finds a BitBake build environment setup script (which defaults to *oe-init-build-env*, but can be configured using property *detect.bitbake.build.env.name*) and at least one package (target image) name is provided using property *detect.bitbake.package.names*.

If you are excluding build dependencies using the *detect.bitbake.dependency.types.excluded* property, the {builddir}/tmp directory must be left intact since a file (license.manifest) that Synopsys Detect uses in that scenario resides in that tmp directory.

Processing

The BitBake detector builds a dependency graph (codelocation) for each given package (target image). It sources your project's build environment setup script (by default: oe-init-build-env), and executes BitBake commands to collect project and dependency details.

The BitBake detector generates one codelocation for each given package (target image) name by performing the following steps:

1. Determines the build directory path by sourcing the given build environment setup script and determining the resulting working directory.
2. Runs 'bitbake --environment' to determine the currently-configured target machine architecture and licenses directory path.
3. Runs 'bitbake-layers show-recipes' to derive the list of layers and collect recipe layer information.
4. For each given package (target image) name:

- If the user requested that build dependencies be excluded, Synopsys Detect locates and reads the license.manifest file for the given package (target image) and the currently-configured target machine architecture. This provides a list of recipes that are included in the target image (the non-build dependencies).
- Runs 'bitbake -g {package}' to generate task-depends.dot, and reads recipes and dependency relationships from it.
- If the user requested that build dependencies be excluded: Synopsys Detect excludes recipes not declared in license.manifest, as well as native recipes. Synopsys Detect always excludes virtual recipes (recipes with names prefixed with "virtual/").
- Synopsys Detect adds at the root level of the graph for the package (target image) each recipe found in task-depends.dot that is not excluded as described above.
- Child (transitive) relationships are created from those root dependencies to their children (as specified in task-depends.dot).

Before running each BitBake command, Synopsys Detect sources the build environment init script, passing any arguments the user has provided via the *detect.bitbake.source.arguments* property.

Configuration

Synopsys Detect properties provide a number of different ways to customize the BitBake detector's behavior for your project. A few of the most important are:

1. You can configure the build environment setup script name using the *detect.bitbake.build.env.name property*.
2. You can add arguments (such as the path to your build directory) to the 'source {build env setup script}' command that Synopsys Detect executes using the *detect.bitbake.source.arguments* property.
3. You can exclude build dependencies from results using the *detect.bitbake.dependency.types.excluded* property.

See the BitBake properties page for a complete list of BitBake detector-related properties and details on how to use them.

Troubleshooting Tips

Missing components for projects using the Yocto Package Revision Service

Symptom: Components are missing from the Black Duck BOM.

Problem: The Yocto Package Revision Service can increment a package revision to a value not present in the Black Duck Knowledge Base, causing a package to fail to match.

Cargo support

Overview

Synopsys Detect runs the Cargo detector if it finds either of the following files in your project:

- Cargo.toml
- Cargo.lock

The Cargo detector parses the Cargo.lock file for information on your project's dependencies. If the detector discovers a Cargo.toml file but not a Cargo.lock file, it will prompt the user to generate a Cargo.lock by running `cargo generate-lockfile` and then run Synopsys Detect again. The Cargo detector extracts the project's name and version from the Cargo.toml file. If it does not find a Cargo.toml file, it will defer to values derived by Git, from the project's directory, or defaults.

Carthage support

Overview

Synopsys Detect runs the Carthage detector if it finds either of the following files in your project:

- Cartfile
- Cartfile.resolved

The Carthage detector parses the Cartfile.resolved file for information on your project's dependencies. If the detector discovers a Cartfile file but not a Cartfile.resolved file, it will prompt the user to generate a Cartfile.resolved file by running `carthage update` and then run Synopsys Detect again.

Supported Dependency Origins

Synopsys Detect only reports dependencies declared in a Cartfile.resolved file that have a 'github' *origin*. This limited support is a result of the de-centralized nature of the Carthage ecosystem. Many commonly-used frameworks used in Carthage projects are not open-source, and thus not tracked by Black Duck.

- Note: Even for dependencies from Github that are declared with the 'github' origin, it is possible that some may not be matched by Black Duck, as Black Duck does not track all repositories hosted on GitHub.

C/C++ (Clang) support

Overview

C/C++ (Clang) support is limited to Linux systems that support one of the following package manager commands: APK, DPKG, or RPM.

For C/C++ projects on Linux systems that meet these requirements, Synopsys Detect can derive dependency information using information read from a JSON Compilation Database (compile_commands.json file) and the Linux package manager. Synopsys Detect will only find components that have been installed into the Linux package manager database.

The JSON compilation database must be generated by your project build before running Synopsys Detect. There are multiple C/C++ build tools that are capable of generating a JSON compilation database. For example, some versions of CMake generate a JSON compilation database when run with the following option:

-DCMAKE_EXPORT_COMPILE_COMMANDS=ON

The Clang detector runs when it finds a compile_commands.json file in the project directory. If the compile_commands.json file resides in a sub-directory, adjust the *detector search depth* to enable Synopsys Detect to find it. Make sure the directory in which the compile_commands.json is located is not being *excluded from detector search*.

For each compile command in the compile_commands.json file, the Clang detector runs a version of the command that is (slightly) modified to ensure that it does not overwrite build artifacts, and generate a list of dependency files used. This is performed by adding the *-M* and *-MF* compiler options. It then uses the Linux package manager to identify which installed package owns each dependency file. These packages are added as a component to the results.

Any dependency file that is not recognized by the Linux package manager and resides outside the source directory (the directory containing the compile_commands.json file) is written to the status.json file under unrecognizedPaths.CLANG.

Conan support

Related properties

Detector properties

Overview

Synopsys Detect has two detectors for Conan:

- Conan Lockfile detector
- Conan CLI detector

Conan detector requirements

Synopsys Detect will run a Conan Detector if either of the following is true:

- Synopsys Detect finds or is provided (via the *detect.conan.lockfile.path* property) a Conan lockfile. (If no lockfile is provided, Synopsys Detect looks for a file named conan.lock.) In this case, the Conan Lockfile detector runs and discovers dependency details using the contents of the Conan lockfile. The Conan Lockfile detector should always be preferred due to the additional information (package revisions) that may be provided by lockfiles.
- Synopsys Detect finds a file named *conanfile.txt* or *conanfile.py*. In this case, the Conan CLI detector runs and discovers dependency details by running the *conan info* command on the Conan project and parsing the output.

In order for Synopsys Detect to generate dependency details that will reliably match components in the Black Duck KnowledgeBase, the Conan revisions feature must be enabled on the Conan project. The Conan command *conan config get general.revisions_enabled* must produce a value of "True" *and* this value must not be overridden by the environment variable CONAN_REVISIONS_ENABLED.

Conan detector usage

When using the Conan CLI detector, be sure to use the *detect.conan.arguments* property to provide any additional arguments (profile settings, etc.) that the *conan info* command needs to produce accurate results.

KB external ID generation

By default (property *detect.conan.attempt.package.revision.match* is set to false), the Conan detectors use the following dependency details to match components in the Black Duck KnowledgeBase:

- name
- version
- user (defaults to "_")
- channel (defaults to "_")
- recipe_revision

For example, here is a conan.lock file entry for a component (zlib):

```
"2": {
 "ref": "zlib/1.2.11#1a67b713610ae745694aa4df1725451d",
 "options": "fPIC=True\nminizip=False\nshared=False",
 "package_id": "d50a0d523d98c15bb147b18fa7d203887c38be8b",
 "prev": "da65bb160c07195dba18afb91259050d",
 "context": "host"
},
```

If you are using the Conan CLI detector instead of the Conan Lockfile detector, this data is found in the output of the `conan info` command instead of the conan.lock file..

The format of the Conan "ref" field is: `<name>/<version>@<user>/<channel>#<recipe_revision>`

In the zlib example:

- name=zlib
- version=1.2.11
- user=_ (by default)
- channel=_ (by default)
- recipe_revision=1a67b713610ae745694aa4df1725451d

Synopsys Detect constructs a KB external ID for namespace "conan" using these fields as follows:

```
<name>/<version>@<user>/<channel>#<recipe_revision>
```

Package revision matching

For Conan projects with lockfiles and the Conan revisions feature enabled Synopsys Detect has an alternative mode, package revision matching, that includes the package ID and package revision in the KB external IDs that it constructs (in addition to the fields described above). (Package revision is provided by Conan lockfiles when the Conan revisions feature is enabled, but it is never provided by the *conan info* command, so this only affects the Conan Lockfile detector.) To enable package revision matching, set property *detect.conan.attempt.package.revision.match* to true.

In this scenario, Synopsys Detect constructs a KB external ID for namespace "conan" as follows:

```
<name>/<version>@<user>/<channel>#<recipe_revision>:<package_id>#<packa
ge_revision>
```

For the zlib example above the two additional fields used in the KB external ID would be:

- package_id=d50a0d523d98c15bb147b18fa7d203887c38be8b
- package_revision=da65bb160c07195dba18afb91259050d

Conan Detector Precedence

If a Conan lockfile (conan.lock) is found or provided, the Conan Lockfile detector will run.

If no Conan lockfile is found or provided, but a conanfile.txt or conanfile.py is found, the Conan CLI detector will run.

Customized user/channel values

Some Conan users use modified versions of Open Source packages with custom user/channel values. These modified components will not match components in the KB. The KB requires a match on user and channel.

Conda Support

Related properties

Detector properties

Overview

The Conda detector discovers dependencies of python projects utilizing the Conda package and environment manager.

Synopsys Detect runs the Conda detector if an environment.yml file is found in your project.

The Conda detector requires that the *conda* executable is on the PATH, or that its path is passed in via `--detect.conda.path`.

The Conda detector runs `conda list -n [environment_name] --json` and `conda info --json`, and parses the output of both commands to discover dependencies.

Note: To specify a Conda environment to be referenced when running `conda list`, pass the name of the environment using `--detect.conda.environment.name` (if not passed, `-n` flag is omitted). Refer to *Properties* for details.

Dart Support

Related properties

Detector properties

Overview

Synopsys Detect has two detectors for Dart:

- Dart CLI detector
- Dart PubSpec Lock detector

Both detectors will run if they find the following files:

- pubspec.yaml
- pubspeck.lock

If Synopsys Detect cannot find a pubspec.lock file, but it finds a pubspec.yaml file, it will prompt the user to run the 'pub get' command to generate the pubspec.lock file, and then run Synopsys Detect again.

Both detectors parse the pubspec.yaml file to determine project name and version information.

The Dart PubSpec Lock detector parses the pubspec.lock file for dependency information. Because the file does not indicate relationships between components, results from this detector will be less accurate than those from the Dart CLI detector.

The Dart CLI detector runs the command 'pub deps' (which requires a pubspec.lock file to be present), and then parses the command's output for dependency information. The detector will first try to run the command using a dart executable (if found), but if it is unsuccessful because the target project requires the Flutter SDK then it will try using a flutter executable (if found).

You may specify the location of dart and flutter executables using the *detect.dart.path* and *detect.flutter.path* properties, respectively.

If you wish to exclude dev dependencies, you may do so using the *detect.pub.dependency.types.excluded* property, which will cause the detector to pass the --no-dev option when running 'pub deps'.

Docker image support

Related properties

Detector properties

Overview

On Linux, Mac, and Windows 10 Enterprise, Synopsys Detect can invoke Detect Docker Inspector to inspect Linux Docker images to discover packages installed by the Linux package manager. For simple use cases, add `--detect.docker.image={repo}:{tag}`, `--detect.docker.tar={path to an image archive}`, or `--detect.docker.image.id={image id}`, to the Synopsys Detect command line.

When passed a value for *detect.docker.image*, *detect.docker.image.id*, or *detect.docker.tar*, Synopsys Detect runs Detect Docker Inspector on the given image (the "target image"), creating one BDIO file for one code location.

Detect Docker Inspector will:

1. Discover packages (components) installed in a given Linux image by analyzing the contents of the Linux package manager database.
2. Provide to Synopsys Detect, for any image, potentially useful targets (file archives) for signature and binary scanning.

Detect Docker Inspector does not run the target image, so it is safe to run it on untrusted images.

While earlier versions of Detect Docker Inspector could be run standalone, the only way to use Detect Docker Inspector now and in the future is to run Synopsys Detect on a Docker image.

Package (component) discovery

For package discovery on a Linux image, Detect Docker Inspector extracts the Linux package manager database from the image, and utilizes the appropriate Linux package manager to provide a

list of the installed packages, which it returns to Synopsys Detect in BDIO (Black Duck Input Output) format. Because it relies on the Linux package manager as its source of this data, the discovered packages are limited to those installed and managed using the Linux package manager.

Detect Docker Inspector can discover package manager-installed components in Linux Docker images that use the DPKG, RPM, or APK package manager database formats.

Signature and binary scan targets

Signature and binary scan targets contain the container file system. The container file system is the file system that a container created from the target image. The container file system is (by default) returned to Synopsys Detect in two forms: as an archive file that contains the container file system (the preferred format for binary scanning), and as a saved squashed (single layer) image that contains the container file system (the preferred format for signature scanning).

Non-linux images

When run on a non-Linux image (for example, a Windows image, or an image that contains no operating system), Detect Docker Inspector will return to Synopsys Detect a BDIO file with zero components along with the signature and binary scan targets. Components may be discovered for these images during the signature and/or binary scanning perfomed by Synopsys Detect.

Modes of operation

Detect Docker Inspector has two modes:

- Host mode, for running on a server or virtual machine (VM) where Detect Docker Inspector can perform Docker operations using a Docker Engine.
- Container mode, for running inside a container started by Docker, Kubernetes, OpenShift, and others.

In either mode, Detect Docker Inspector runs as a Synopsys Detect inspector to extend the capaibilities of Synopsys Detect. Detect Docker Inspector is more complex than most Synopsys Detect inspectors because it relies on container-based services (the image inspector services) to perform its job. When running on a host machine that has access to a Docker Engine ("host mode"), Detect Docker Inspector can start and manage the image inspector services (containers) automatically. When Synopsys Detect and Detect Docker Inspector are running within a Docker container ("container mode"), the image inspector services must be started and managed by the user or the container orchestration system.

Host mode

Host mode (the default) is for servers/VMs where Detect Docker Inspector can perform Docker operations (such as pulling an image) using a Docker Engine.

Host mode requires that Detect Docker Inspector can access a Docker Engine. https://github.com/docker-java/docker-java utilizes the *docker-java library* to act as a client of that Docker Engine. This enables Detect Docker Inspector to pull the target image from a Docker registry such as Docker Hub. Alternatively, you can save an image to a .tar file by using the *docker save* command. Then, run Detect Docker Inspector (via Synopsys Detect) on the .tar file. See *Supported image formats* for details on supported .tar file formats.

In Host mode, Detect Docker Inspector can also pull, run, stop, and remove the image inspector service images as needed, greatly simplifying usage, and greatly increasing run time.

Container mode

Container mode is for container orchestration environments (Kubernetes, OpenShift, etc.) where Synopsys Detect and Detect Docker Inspector run inside a container where Detect Docker Inspector cannot perform Docker operations. For information on running Detect Docker Inspector in container mode, refer to *Deploying*.

It is possible to utilize container mode when running Synopsys Detect and Detect Docker Inspector on a host that supports host mode. Container mode is more difficult to manage than host mode, but you might choose container mode in order to increase throughput (to scan more images per hour). Most of the time spent by Detect Docker Inspector running in host mode is spent starting and stopping the image inspector services. When these services are already running (in the usual sense of the word "service") as they do in container mode, Detect Docker Inspector executes much more quickly than it would in host mode.

Requirements

Requirements for including Detect Docker Inspector in a Synopsys Detect run include of all of Synopsys Detect's requirements plus:

- Three available ports for the image inspector services. By default, these ports are 9000, 9001, and 9002.
- The environment must be configured so that files created by Detect Docker Inspector are readable by all. On Linux, this means an appropriate umask value (for example, 002 or 022 would work). On Windows, this means the Detect "output" directory (controlled by the Synopsys Detect property *detect.output.path*) must be readable by all.
- In host mode: access to a Docker Engine versions 17.09 or higher running as root.
- In container mode: you must start the Detect Docker Inspector container that meets the preceding requirements, and three container-based "image inspector" services. All four of these containers

must share a mounted volume and be able to reach each other through HTTP GET operations using base URLs that you provide. For more information, refer to *Deploying*.

Running Detect Docker Inspector

To invoke Detect Docker Inspector, pass a docker image to Synopsys Detect via one of the following properties:

- detect.docker.image
- detect.docker.image.id
- detect.docker.image
- detect.docker.tar

See the Synopsys Detect documentation for details.

Advanced usage (using passthrough properties)

The most common cases of Detect Docker Inspector can be configured using Synopsys Detect properties. However, there are scenarios (including container mode) that require access to Detect Docker Inspector advanced properties for which there is no corresponding Synopsys Detect property. For the list of Detect Docker Inspector advanced properties, see *Advanced properties*.

When you need to set one of the Detect Docker Inspector advanced properties, construct the Synopsys Detect property name by prefixing the Docker Inspector property name with `detect.docker.passthrough.`.

Suppose you need to set Docker Inspector's `service.timeout` value (the length of time Docker Inspector waits for a response from the Image Inspector services that it uses) to 480000 milliseconds. You add the prefix to the Docker Inspector property name to derive the Synopsys Detect property name `detect.docker.passthrough.service.timeout`. Therefore, add `--detect.docker.passthrough.service.timeout=480000` to the Synopsys Detect command line.

For example:

```
./detect8.sh --detect.docker.image=ubuntu:latest --detect.docker.passth
rough.service.timeout=480000
```

You can set any Detect Docker Inspector property using this method. However, you should not use this method to change the value of the Detect Docker Inspector property `output.path`. Synopsys Detect sets this property and changing its value via the passthrough mechanism will make it impossible for Synopsys Detect to find Detect Docker Inspector's output files.

Transitioning from Black Duck Docker Inspector to Synopsys Detect

If you have been running the Black Duck Docker Inspector directly, and need to transition to invoking Detect Docker Inspector from Synopsys Detect, here are some recommendations likely to help you make the transition:

1. If you run Black Duck Docker Inspector with `blackduck-docker-inspector.sh`, replace `blackduck-docker-inspector.sh` in your command line with `detect8.sh` (adjust the Synopsys Detect major version as necessary). See the Synopsys Detect documentation for information on where to get the Synopsys Detect script.
2. If you run Black Duck Docker Inspector with `java -jar blackduck-docker-inspector-{version}.jar`, replace `blackduck-docker-inspector-{version}.jar` in your command line with `synopsys-detect-{version}.jar`. See the Synopsys Detect documentation for information on where to get the Synopsys Detect .jar.
3. For each of the following properties used in your command line, add `detect.` to the beginning of the property name: docker.image, docker.image.id, docker.tar, docker.platform.top.layer.id. For example, change `--docker.image=repo:tag` with `--detect.docker.image=repo:tag`.
4. For all other Docker Inspector properties used in your command line, add `detect.docker.passthrough.` to the beginning of the property name. For example, change `--bdio.organize.components.by.layer=true` to `--detect.docker.passthrough.bdio.organize.components.by.layer=true`.

Supported image formats

Images passed to Synopsys Detect via the *detect.docker.image* property must either be pullable using the machine's docker engine (via the equivalent of a "docker pull" command) or already exist in the local docker cache. Synopsys Detect will save these to a file using the equivalent of a "docker save" command.

Images passed to Synopsys Detect via the *detect.docker.image.id* property must already exist in the local docker cache. Synopsys Detect will save these to a file using the equivalent of a "docker save" command.

Image files passed to Synopsys Detect via the *detect.docker.tar* property must be .tar files, and the contents must conform to either of the following image format specifications: 1. *Docker Image Specification v1.2.0* (the format produced by the "docker save" command), or 2. *Open Container Initiative Image (OCI) Format Specification*.

Synopsys Detect workflow

When running Synopsys Detect on a Docker image, you'll probably want to set *detect.target.type* to *IMAGE*.

When a Docker image is provided and property *detect.target.type* is set to IMAGE, Synopsys Detect will run:

1. Docker Inspector (on the image)
2. Black Duck Signature Scanner (on the image)
3. Black Duck - Binary Analysis (on the image)

When a Docker Image is provided and property *detect.target.type* is set to *SOURCE* (the default), Synopsys Detect will run:

1. Docker Inspector (on the image)
2. Any applicable detectors (on the source directory)
3. Black Duck Signature Scanner (on the image)
4. Black Duck - Binary Analysis (on the image)

Synopsys Detect by default runs the Black Duck Signature Scanner on an image built from the "container file system". This image is referred to as the squashed image (because it has only one layer, to eliminate the chance of false positives from lower layers). This scan creates another code location.

Synopsys Detect by default runs Black Duck - Binary Analysis on the container file system. Refer to *Synopsys Detect's scan target* for more details. This also creates a code location.

File permissions

When using Docker Inspector, Synopsys Detect must be run in an environment configured so that files created by Docker Inspector are readable by all. On Linux, this means an appropriate umask value (for example, 002 or 022 will work). On Windows, this means that the Synopsys Detect output directory must be readable by all.

Docker image tarfiles passed to Synopsys Detect via the *detect.docker.tar* property must be readable by all.

Synopsys Detect's scan target

When a Docker image is run; for example, using a docker run command, a container is created that has an initial file system. This initial container file system can be determined in advance from the image without running the actual image. Since the target image is not yet trusted, Docker Inspector does not run the image; that is, it does not create a container from the image, but it does construct the initial container file system.

When Synopsys Detect invokes both Black Duck Signature Scanner, and Docker Inspector because detect.docker.image or detect.docker.tar are set, the target of that Black Duck signature scan is the initial container file system constructed by Docker Inspector. The intial container file system is packaged in a way to optimize results from Black Duck's matching algorithms. Rather than directly running the Black Duck Signature Scanner on the initial container file system, Synopsys Detect runs the Black Duck Signature Scanner on a new image; in other words, the squashed image, constructed using the initial container file system built by Docker Inspector. Packaging the initial

container file system in a Docker image triggers matching algorithms within Black Duck that optimize match results for Linux file systems.

By default, Synopsys Detect also runs Black Duck - Binary Analysis on the initial container file system. If your Black Duck server does not have Black Duck - Binary Analysis enabled, you should disable Black Duck - Binary Analysis. For example, you might set: `--detect.tools.excluded=BINARY_SCAN`.

Isolating application components

If you are interested in components from the application layers of your image, but not interested in components from the underlying platform layers, you can exclude components from platform layers from the results.

For example, if you build your application on ubuntu:latest (your Dockerfile starts with FROM ubuntu:latest), you can exclude components from the Ubuntu layer(s) so that the components generated by Synopsys Detect using Docker Inspector and the Black Duck Signature Scanner contain only components from your application layers.

First, find the layer ID of the platform's top layer. To do this task:

Run the `docker inspect` command on the base image; in our example this is *ubuntu:latest*. Find the last element in the *RootFS.Layers* array. This is the platform top layer ID. In the following example, the platform top layer ID is *sha256:b079b3fa8d1b4b30a71a6e81763ed3da1327abaf0680ed3ed9f00ad1d5de5e7c*. Set the value of the Docker Inspector property docker.platform.top.layer.id to the platform top layer ID. For example:

```
./detect8.sh ... --detect.docker.image={your application image} --detec
t.docker.platform.top.layer.id=sha256:b079b3fa8d1b4b30a71a6e81763ed3da1
327abaf0680ed3ed9f00ad1d5de5e7c
```

In this mode, there may be some loss in match accuracy from the Black Duck Signature Scanner because, in this scenario, the Black Duck Signature Scanner may be deprived of some contextual information, such as the operating system files that enable it to determine the Linux distribution, and that that may negatively affect its ability to accurately identify components.

Inspecting Windows Docker images

Given a Windows Image, Docker Inspector, since it can only discover packages using a Linux package manager, will not contribute any components to the BOM, but will return the container filesystem (in the form of a squashed image), which Synopsys Detect will scan using the Black Duck Signature Scanner.

Inspecting Docker images on Windows

This section contains considerations that apply when running on Windows.

Docker usually needs to be configured for Linux containers

By default, Docker Inspector pulls (using Docker) and uses container-based image inspector services. These services run on Linux operating systems. On Windows, Docker must be configured for Linux containers in order for these Linux-based images to be pulled and started by Docker Inspector.

Docker Inspector also supports more advanced deployment options that avoid this requirement by using image inspector services running elsewhere in the network; for more information, refer to the Docker Inspector documentation.

Docker must be able to mount the shared directory as a volume

Docker Inspector uses Docker to mount a shared directory (a directory inside Synopsys Detect's output directory, which is controlled via property detect.output.path) so that it can be accessed (read from and written to) by the image inspector container. Consequently Docker must have permissions sufficient to mount read/write directories within Synopsys Detect's output directory.

The user must have permission to create symbolic links

Docker Inspector constructs the initial file system that a container would have if you ran the target image. All component discovery (including package manager discovery and signature scanning) is based on this file system. In order to construct the file system accurately, the Synopsys Detect user must have permission to create symbolic links.

A Docker bug may affect Synopsys Detect during clean up

For important information on a Docker for Windows bug that might affect Synopsys Detect, refer to the *troubleshooting page*.

Architecture overview

Detect Docker Inspector uses up to three container-based image inspector services; one for each of the supported Linux package manager database formats.

The three image inspector services provide coverage of the three package manager database formats: DPKG, RPM, and APK. By default, Detect Docker Inspector submits its request to inspect the target image to the DPKG (Ubuntu) image inspector service. All services redirect to the appropriate image inspector service if it cannot handle the request. For example, if the target image is a Red Hat image, the Ubuntu inspector service, which cannot inspect a Red Hat image, redirects to the CentOS inspector service, which can inspect a Red Hat image. If you know that most of your images have either RPM or APK databases, you can improve performance by configuring Detect

Docker Inspector to send requests to the CentOS (RPM) or Alpine (APK) image inspector service using the Detect Docker Inspector property *imageinspector.service.distro.default*.

In host mode (the default), Detect Docker Inspector automatically uses the Docker engine to pull as needed from Docker Hub the following three images: blackducksoftware/blackduck-imageinspector-alpine, blackducksoftware/blackduck-imageinspector-centos, and blackducksoftware/blackduck-imageinspector-ubuntu. Detect Docker Inspector starts those services as needed, and stops and removes the containers when Detect Docker Inspector exits. It uses a shared volume to share files, such as the target Docker image, between the Detect Docker Inspector utility and the three service containers.

In container mode, start the container running Detect Docker Inspector and the three image inspector container-based services such that all four containers share a mounted volume and can communicate with each other using HTTP GET operations using base URLs that you provide. For more information, refer to *Deploying*.

Execution modes

Host mode

In host mode, Detect Docker Inspector performs the following steps on the host:

1. Pulls and saves the target image to a .tar file if you passed the image by *repo:tag*.
2. Checks to see if the default image inspector service is running. If not, it pulls the inspector image and starts a container, mounting a shared volume.
3. Requests the Black Duck input/output (BDIO) file and container file system by sending an HTTP GET request to the image inspector service.

The following steps are performed inside the image inspector container:

1. Builds the container file system that a container would have if you ran the target image. It does not run the target image.
2. Determines the target image package manager database format, and redirects to a different image inspector service if necessary.
3. Runs the image inspector's Linux package manager on the target image package manager database to get details of installed packages.
4. Produces and returns a BDIO1 (.jsonld) file consisting of a graph of target image packages and, optionally, the container filesystem.

The following steps are performed back on the host when the request to the image inspector service returns:

1. Returns the output files (BDIO and signature and binary scan targets) to Synopsys Detect by copying them to the output directory.
2. Stops/removes the image inspector container. Note that this can be disabled.

Container mode

In container mode, you start four containers in such a way that they share a mounted volume and can reach each other through HTTP GET operations using base URLs that you provide:

- One container for Synopsys Detect / Detect Docker Inspector.
- One container for each of the three image inspector services: Alpine, CentOS, and Ubuntu.

In container mode you must provide the target image in a .tar file with one of the supported formats; you cannot specify that target image by repo:tag.

Synopsys Detect invokes Detect Docker Inspector, which requests the dependency graph (in BDIO format) and signature/binary scan targets using HTTP from the default image inspector service using a base URL that you have provided.

The following steps are performed inside the image inspector container:

1. Builds the container file system that a container would have if you ran the target image. It does not run the target image.
2. Determines the target image package manager database format, and redirects to a different image inspector service if necessary.
3. Runs the image inspector's Linux package manager on the target image package manager database to get details of the installed packages.
4. Produces and returns a BDIO1 (.jsonld) file consisting of a graph of target image packages and, optionally, the container filesystem.

The following steps are performed by Detect Docker Inspector/Synopsys Detect back in the Synopsys Detect container when the request to the image inspector service returns:

1. Detect Docker Inspector returns the output files (BDIO and signature and binary scan targets) to Synopsys Detect by copying them to the output directory.
2. Synopsys Detect converts the BDIO to BDIO2, adjusts the project, project version, and codelocation names, and uploads it to Black Duck.
3. Synopsys Detect performs Black Duck signature scan and Black Duck - Binary Analysis on the scan targets.

Deploying in container mode

Deploying in container mode is challenging and requires expertise in the container platform on which you will deploy. We recommend engaging Synopsys Synopsys Software Integrity Group Client Services for a solution tailored to your environment.

You can find several container mode deployment examples on the *deployment page*.

Advanced topics

This section covers a variety of advanced topics.

How Detect Docker Inspector discovers dependencies

Detect Docker Inspector discovers dependencies in the target image by making a request to an image inspector service (running inside a container). The image inspector service works as follows:

1. Reads the target image, and constructs the file system that the container would have at time zero if you were to run the image.
2. Finds the linux package manager database in that file system.
3. Runs its own linux package manager (invoking its "list" function: "dpkg -l", "rpm -qa --qf ...", or "apk info -v") on the package manager database from the target image to generate the list of packages installed in the target image. If the package manager database type of the target image does not match the image inspector's package manager type, the image inspector will redirect the request to the image inspector that can process the target image (which starts again at step 1).
4. Translates that package list to BDIO and uploads it to Black Duck, or returns it to the the caller (e.g. Synopsys Detect) to upload to Black Duck.
5. Returns the constructed file system to the caller (e.g. Synopsys Detect) for signature scanning.

The user ID and group ID of the image inspector service process will (in general) be different from the user ID and group ID of the Detect Docker Inspector process. Consequently, the the environment must be confugred so that files created by Detect Docker Inspector are readable by all. On Linux, this means an appropriate umask value (for example 002 or 022). On Windows, the working directory must be readable by all. In addition, the permissions on a Docker tarfile passed via the *docker.tar* property must allow read by all.

Enabling file sharing in Docker

Docker may restrict file sharing to certain directories that you've configured in Docker > Settings > Resource > File Sharing. To enable Detect Docker Inspector to mount the volumes that it needs, you may need to enabling file sharing for directory $HOME/blackduck.

Organizing components by layer

By default Detect Docker Inspector will produce BDIO containing a component graph consisting only of components (linux packages).

Alternatively, you can direct Detect Docker Inspector to organize components by image layer by setting property `bdio.organize.components.by.layer=true`. Run this way, Detect Docker Inspector will produce BDIO containing image layers at the top level of the graph, and components

associated with each layer appearing as children of that layer. This structure is visible in from the Black Duck project version Source display.

A side effect of this components-under-layers graph structure is the categorization by Black Duck of all components as Transitive.

In the BDIO, layers are named `Layer{index}_{layer digest}`, where `{index}` is a two digit index starting at 00 to indicate layer ordering within the image, and `{layer digest}` is the layer digest with ":" replaced with "_". For example, the first layer of an image could be named: `Layer00_sha256_1bcfbfaf95f95ea8a28711c83085dbbeceefa11576e1c889304aa5bacbaa6ac2`.

Because this feature produces BDIO in which the same component may appear at multiple points in the graph, only Black Duck versions 2021.8.0 and newer have the ability to correctly display graphs organized by layer, and only if *Admin* > *System Settings* > *Scan* > *Component Dependency Duplication Sensitivity* is set high enough to avoid removal of components that appear multiple times in the graph (at minimum: 2).

When organizing components by layer, you must also choose whether or not to include removed (whited-out) components in the output.

Including removed components

If you include removed components, Detect Docker Inspector will produce BDIO containing a graph with image layers at the top level. Under each image layer it will include all components present after the layer was applied to the file system (including components added by lower layers that are still present). If a component is added by a lower layer but later removed (whited-out) by a higher layer, it will not appear under the layer that removed it or any higher layer (unless/until it is re-added by another layer).

The benefit of using this mode: for every component added by any layer, you can see where it was added, and where it was removed (if it was).

The downside of using this mode: components added by a lower layer but removed by a higher layer will appear in the BOM, even though they are not present in the final container filesystem.

To include removed components, set property `bdio.include.removed.components=true`.

Excluding removed components

If you exclude removed components (the default), Detect Docker Inspector will behave as described in the section above except that components not present in the final container filesystem will not appear on any layer in the BDIO graph.

The benefit of using this mode: components added by a lower layer but removed by a higher layer (and therefore not present in the final container filesystem) will not appear in the BOM.

The downside of using this mode: if a component is added by a lower layer and removed by a higher layer, there is no evidence of that in the BDIO graph or the BOM.

To exclude removed components, set property `bdio.include.removed.components=false` (the default).

Isolating application components

If you are interested in components from the application layers of your image, but not interested in components from the underlying platform layers, you can exclude components from platform layers from your results.

For example, if you build your application on *ubuntu:latest* (your Dockerfile starts with FROM ubuntu:latest), you can exclude components from the Ubuntu layer(s) so that the components generated by Detect Docker Inspector contain only components from your application layers.

Limitations to this feature

This feature works when the method you use to build your image adds layers on top of a base or platform image (single-stage builds). Here's an example of a Dockerfile that adds layers on top of a base image. This feature will work for images built this way:

```
FROM ubuntu:latest
RUN apt-get update && apt-get -y install curl
```

The resulting image will include all layers from ubuntu:latest, plus an additional layer that adds the curl package. This feature is designed for this type of image, and will be able to isolate the layers added to ubuntu:latest (which in this case is a single layer that adds curl).

On the other hand, when you build an image using multiple FROM statements in the Dockerfile (multi-stage builds), the layers in the resulting image are not, in general, the layers of the first image followed by the layers of the second image. You can verify this by comparing the *RootFS.Layers* list of the resulting image to the *RootFS.Layers* lists of the images named in the FROM statements. (See below for instructions on how to get the RootFS.Layers list using the *docker inspect* command).

Here's an example of a Dockerfile that uses multiple FROM statements. This feature will not, in general, work for images built this way:

```
FROM ubuntu:latest
FROM myapplicationimage:mytag
```

How to use this feature

First, find the layer ID of the platform's top layer using the following process.

1. Run the *docker inspect* command on the platform image. In this example, the platform image is ubuntu:latest, so you would run `docker inspect ubuntu:latest`.
2. Find the last element in the *RootFS.Layers* array. This is the platform top layer ID. In the following example, this is

sha256:b079b3fa8d1b4b30a71a6e81763ed3da1327abaf0680ed3ed9f00ad1d5de5e7c. (Because the *latest* tag moves frequently, the top layer ID for ubuntu:latest changes over time.)

Set the value of the Synopsys Detect property *detect.docker.platform.top.layer.id* to the platform top layer ID. For example:

```
{[solution_name] command} ... --docker.platform.top.layer.id=sha256:b07
9b3fa8d1b4b30a71a6e81763ed3da1327abaf0680ed3ed9f00ad1d5de5e7c
```

In this mode, the container file system and/or container file system squashed image produced by Detect Docker Inspector only contains files added to the image by application layers. If the Black Duck signature scanner is run on this file, it generates results based only on files found on application layers. This provides the benefit of isolating application components by excluding platform components. However, there may be some loss in match accuracy from the signature scanner because in this scenario, the signature scanner may be deprived of some contextual information, such as the operating system files that enable it to determine the Linux distribution.This could negatively affect its ability to accurately identify components.

Considerations when running on Windows

Docker file sharing settings

Detect Docker Inspector requires the ability to share directories with the image inspector containers. It shares a directory with image inspector containers by mounting it as a volume. You will need to configure your Docker settings to enable this file sharing. The simplest way to do this is to add your home directory as a sharable directory on the Docker settings Resources > FILE SHARING screen.

The shared directories are created under the Synopsys Detect output directory (controlled by Synopsys Detect *detect.output.path*). If you change the location of the Synopsys Detect output directory, be sure your Docker file sharing settings enable sharing of that directory.

Docker restrictions

Docker on Windows has restrictions that impact Detect Docker Inspector:

1. Docker can be configured to pull either Linix images, or Windows images. You can see how your Docker installation is configured by looking at the *OSType* value in the output of the *docker info* command. If Docker is configured for Linix images, it cannot pull Windows images, and vice versa. The command to change Docker's *OSType* value appears in the Docker Desktop menu. Refer to Docker documentation for more information.
2. When pulling Windows images, Docker requires (a) that the architecture of the pulled image matches the architecture of your machine, and (b) that the Windows version of the pulled image is a close match to the Windows version of your machine.

Inspecting Windows images on non-Windows systems

Running on a non-Windows system, Docker can neither pull nor build a Windows image. Consequently, you cannot run Detect Docker Inspector on a Windows Docker image using the either the *docker.image* or *docker.image.id* property. Instead, you must, using a Windows system, pull and save the target image as a .tar file, and pass that .tar file to Detect Docker Inspector using the *docker.tar* property.

Inspecting multiple images more efficiently by leaving services running

By default, Detect Docker Inspector starts, uses, and then stops and removes either one or two containerized image inspector services per run. This may be appropriate when scanning a single image, but when scanning many images it is highly inefficient, and it doesn't support concurrent execution of multiple Detect Docker Inspector runs.

The scanning of many images is completed significantly faster by starting the image inspector services once, and running multiple instances of Detect Docker Inspector, so that each one sends requests to the already-running image inspector services.

The following script illustrates how this is done in a Docker environment:

```
curl -O [source_raw_content_url_base]/[source_repo_organization]/[proje
ct_name]/master/deployment/docker/batchedImageInspection.sh
```

To keep the example simple, this script only starts the Alpine image inspector service. In general, you must start two more services: the Ubuntu image inspector service for inspecting images built from dpkg-based Linux distros, and the CentOS image inspector service for inspecting images built from rpm-based Linux distributions. It doesn't matter which service receives the request; any service redirects if necessary.

Configuring Detect Docker Inspector for your Docker Engine and registry

If you invoke Detect Docker Inspector with an image reference; in other words, a repo:tag value versus a .tar file, it uses the *docker-java library*, to access the Docker registry to pull the image.

If you can pull an image using *docker pull* from the command line, then you will be able to configure Detect Docker Inspector to pull that image. The docker-java library can often be configured the same way as the Docker command line utility (*docker*).

There are also other ways to configure docker-java. For more information on configuring docker-java (and therefore Detect Docker Inspector) for your Docker registry, refer to the configuration information in the *docker-java documentation*.

For example, one way to configure your Docker registry username and password is to create a file in your home directory named *.docker-java.properties* that configures username and password:

```
registry.username=mydockerhubusername
registry.password=mydockerhubpassword
```

If you need to override the DOCKER_HOST value, set property *use.platform.default.docker.host* to false. When *use.platform.default.docker.host* is set to false, Detect Docker Inspector does not override any of the configuration settings in the code, so all other docker-java configuration methods such as properties, system properties, and system environment, are available to you.

When *use.platform.default.docker.host* is set to true (the default value) *and* Detect Docker Inspector is running on Windows, Detect Docker Inspector overrides only the DOCKER_HOST value (setting it to "npipe:////./pipe/docker_engine").

Running Synopsys Detect on a project directory that exists within a Docker image

When you want to run Synopsys Detect on a directory that exists within a Docker image, you can use the following approach:

1. Run Synopsys Detect on the image to generate the container filesystem for the image.
2. Run Synopsys Detect on a directory within that container filesystem.

Synopsys Detect performs these actions without running the image/container.

To see a simple example that illustrates this approach, use the following commands to download these sample files:

```
curl -O [source_raw_content_url_base]/[source_repo_organization]/[proje
ct_name]/master/deployment/docker/runDetectInImageDir/runDetectInImageD
ir.sh
curl -O [source_raw_content_url_base]/[source_repo_organization]/[proje
ct_name]/master/deployment/docker/runDetectInImageDir/Dockerfile
```

Review the script before running it to make sure the side effects (files and directories that it creates) are acceptable. You must make the script executable before you run it.

Running the signature scanner on a specific directory within a Docker image

To use iScan to scan a specific directory within an image:

1. Run Detect Docker Inspector on the target image to get the container file system. You can also do this using Synopsys Detect using `--detect.docker.passthrough.*` properties. Include the following Detect Docker Inspector properties:

```
--upload.bdio=false                          # disable BDIO upload
--output.include.containerfilesystem=true  # tell DI to output the cont
ainer file system
--output.path={your output dir}              # tell DI where to put the o
utput
```

1. Locate the container file system in the output directory (*.tar.gz) and untar it.
2. cd into the directory within the untared container file system that you want to scan.
3. Invoke Synopsys Detect there.

Excluding files/directories from the returned container file system which excludes them from Synopsys Detect's signature scan

To exclude certain files and/or directories from the returned file system, you can specify that list of directories with the property *--output.containerfilesystem.excluded.paths*.

For example, if you are invoking Detect Docker Inspector from Synopsys Detect, and want Synopsys Detect to exclude the */etc* and */usr/bin* directories from the signature scan, you could run Synopsys Detect like this:

```
./detect.sh --detect.docker.image=ubuntu:latest --detect.docker.passthr
ough.output.containerfilesystem.excluded.paths=/etc,/usr/bin
```

OCI Image support

Detect Docker Inspector supports *OCI image* archives (.tar files) passed to it via the Synopsys Detect detect.docker.tar property.

Detect Docker Inspector derives the image repo:tag value for each manifest in the index.json file from an annotation with key 'org.opencontainers.image.ref.name' if it is present.

If the OCI archive contains multiple images, Detect Docker Inspector constructs the target repo:tag from the values of Synopsys Detect properties detect.docker.passthrough.docker.image.repo and detect.docker.passthrough.docker.image.tag (if detect.docker.passthrough.docker.image.tag is not set it defaults to "latest"), and looks for a manifest annotation with key 'org.opencontainers.image.ref.name' that has value matching the constructed target repo:tag. If a match is found, Detect Docker Inspector inspects the matching image. If no match is found, or detect.docker.passthrough.docker.image.repo is not set, Detect Docker Inspector fails.

Advanced properties

The table below lists advanced Docker Inspector properties that can be set using the "detect.docker.passthrough." property prefix:

Property name	Type	Description	Default value	Deprecation St	Deprecation M
bdio.include.removed.components	Boolean	In generated BDIO, include removed components? If false, only components present in the final container filesystem (in other words, present after the final layer is applied) will be included in the output. If true, a component added by any layer will be included in the output even if later removed by a higher layer.	false		
bdio.organize.components.by.layer	Boolean	In generated BDIO, organize components by layer?	false		

Property name	Type	Description	Default value	Deprecation St	Deprecation M
cleanup.inspector.container	Boolean	Stop inspector container after using it?	true		
cleanup.inspector.image	Boolean	Remove inspector image after using it?	false		
cleanup.target.image	Boolean	Remove target image after saving it?	false		
cleanup.working.dir	Boolean	Cleanup Working Dir?	true		
command.timeout	Long	Command Timeout (Milliseconds)	120000		

Property name	Type	Description	Default value	Deprecation St	Deprecation M
docker.image.platform	String	The platform (shown as 'platform' field in 'docker manifest inspect {image}' output) of the target image Docker Inspector should pull. You must also provide the target image (via docker.image) when using this property. Note: when providing a platform, you may provide either the target operating system (os), the target architecture, or os/architecture.			
docker.image.repo	String	Docker Image Repo; Use with docker.image.tag to select one image from a tarfile			
docker.image.tag	String	Docker Image Tag; Use with docker.image.repo to select one image from a tarfile			

Property name	Type	Description	Default value	Deprecation St	Deprecation M
imageinspector.service.distro.default	String	Default image inspector Linux distro (alpine, centos, or ubuntu)	ubuntu		
imageinspector.service.log.length	Integer	The number of lines of the image inspector service log to include in the Docker Inspector log when log level is DEBUG or higher	50000		
imageinspector.service.port.alpine	String	alpine image inspector host port	9000		
imageinspector.service.port.centos	String	centos image inspector host port	9001		
imageinspector.service.port.ubuntu	String	ubuntu image inspector host port	9002		
imageinspector.service.start	Boolean	Start ImageInspector services (containers) as needed?	true		
imageinspector.service.url	String	The URL of the (already running) imageinspector service to use			

Property name	Type	Description	Default value	Deprecation St	Deprecation M
linux.distro	String	Target image Linux distribution name. Use this if you need to override the Linux distribution name discovered by Docker Inspector.			
offline.mode	Boolean	Make no attempts to access network-based resources (the docker repository)	false		
output.containerfilesystem.excluded.paths	String	Comma-separated list of directories/files/links (specified as absolute paths) to exclude from the container filesystem			
output.include.containerfilesystem	Boolean	Include container filesystem (a large file) in output?	false		
output.include.squashedimage	Boolean	Include container filesystem (a large file) in output?	false		
output.path	String	Path to directory for output files			

Property name	Type	Description	Default value	Deprecation St	Deprecation M
service.timeout	Long	HTTP Service Request Timeout (Milliseconds)	600000		
shared.dir.path.local	String	The host's path to the dir shared with the imageinspector containers. Only needed if using existing imageinspector containers. If not set, $HOME/blackduck/docker-inspector/shared will be used			
system.properties.path	String	Path to a properties file containing additional system properties (an alternative to java -D)			
use.platform.default.docker.host	Boolean	Use platform's default DOCKER_HOST value? Set to false if you want to override DOCKER_HOST	true		

Deploying Detect Docker Inspector

Detect Docker Inspector can be run in either of the following modes:

1. Host mode on a server or virtual machine (VM) with Docker. In this mode, Detect Docker Inspector starts and stops the image inspector services it uses. The deployment approach for

host mode is referred to below as "utility;" you simply execute a command, and deployment is automatic.

2. Container mode utilizing either Docker or a container running on an orchestration platform such as Kubernetes, OpenShift, among others. In this mode, you start the image inspector services, and Detect Docker Inspector just sends them requests, which complete much faster. The deployment approach for container mode is referred to below as "toolkit;" you take components provided by Docker Inspector (one utility, three containerized services) and deploy them yourself.

Most, but not all, of the following deployment examples use the toolkit approach.

Important notes regarding deployment sample code

The deployment samples provided are intended to illustrate possible approaches to the challenges involved in deploying Detect Docker Inspector. They are not intended to be used as-is in production. You should understand the code before you use it. They do not represent the only way to deploy in each environment. At a minimum, before using a sample, you should update it to the latest versions of the software it uses.

Your deployment approach is the same whether you are invoking Detect Docker Inspector directly, or invoking it using Synopsys Detect. Most of the sample deployments use Synopsys Detect simply because that is the most common use case.

Using host mode on a server or VM with Docker

In this scenario, Detect Docker Inspector is a command line utility that automatically pulls/runs and uses container-based services,and cleans them up when it's done. The Docker command, if installed on the machine, can be very useful for troubleshooting, but is not actually required or used by Detect Docker Inspector.

In this mode, Detect Docker Inspector requires access to a Docker Engine, similar to the way the Docker client requires access to a Docker Engine, so it can pull and run Docker images. It uses the *docker-java* library to perform Docker operations using the Docker Engine.

In this mode, Detect Docker Inspector automatically pulls, runs, stops, and removes the container-based image inspector services on which it depends. It accesses the services they provide through HTTP GET operations.

This is the default mode, and the simplest to use.

Using container mode with Docker or a container orchestration platform such as Kubernetes, OpenShift, and others

In this scenario, Detect Docker Inspector is a toolkit consisting of a command line utility that you run in one container, plus three container-based services which you must start. These four containers must: (a) Share a mounted volume, either persistent or temporary, used to pass large files between

containers, and: (b) Be able to reach each other through HTTP GET operations using base URLs that you provide.

Because in this mode you (not Detect Docker Inspector) are deploying the image inspector services, you must ensure that you deploy the correct version of the image inspector images for the version of Detect Docker Inspector that you run. This is easier if you explicitly control the version of Detect Docker Inspector, rather than letting Synopsys Detect auto-update Detect Docker Inspector. See the Synopsys Detect for details.

Image Inspector Services

Detect Docker Inspector consists of a command line utility provided in a Java .jar, but sometimes invoked using a bash script, and three image inspector services.

The required Docker operations, if any, are performed by the command line utility, while the image inspector services perform the work of unpacking the target Docker image, extracting the Linux package manager database, and running the Linux package manager against that database to extract installed packages and translate them to components which are actually externalIDs for Black Duck. If the image inspector service finds in the target image a package manager database that is incompatible with its own package manager utility; for example, when you run Detect Docker Inspector on an Alpine image, but the request goes to the Ubuntu image inspector service, the image inspector service redirects the request to the appropriate image inspector service. You can change the default image inspector service to reduce the likelihood of redirects, resulting in shorter execution times. For example, if most of your target images are Alpine, you can set *imageinspector.service.distro.default* to *alpine*.

The image inspector service containers are downloaded from Docker Hub (blackducksoftware/blackduck-imageinspector-*).

Deployment sample for Docker using persistent image inspector services

Approach: Toolkit

Deployment notes: Detect Docker Inspector runs on a host that has Docker. Each image inspector service runs in a container. The shared volume is a directory on the host, mounted into each container. The containers communicate through service URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/docker/runDetectAgainstDockerServices/setup.sh

Deployment sample for Kubernetes

Approach: Toolkit

Deployment notes: Each image inspector service runs in a separate pod. The shared volume is a hostPath volume. The containers communicate through service URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/kubernetes/setup.txt

Deployment sample for OpenShift

Approach: Toolkit

Deployment notes: All image inspector services run in a single pod. The shared volume is an *emptyDir* volume. The containers communicate through localhost URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/openshift/setup.txt

Deployment sample for Travis CI

Approach: Toolkit

Deployment notes: Uses the Travis CI Docker service. The containers communicate through localhost URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/travisci/travis.yml

Deployment sample for GitLab CI

Approach: Toolkit

Deployment notes: Uses the GitLab CI shell executor. The containers communicate through localhost URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/gitlabci/setup.sh

Deployment sample for Circle CI

Approach: Utility

Deployment notes:

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/circleci/config.yml

Deployment sample for Docker with Synopsys Detect running in a container

Approach: Toolkit

Deployment notes: The containers communicate through localhost URLs.

Download: curl -O https://raw.githubusercontent.com/blackducksoftware/detect-docker-inspector/master/deployment/docker/runDetectInContainer/setup.sh

Configuring Detect Docker Inspector for your Docker registry

If you invoke Detect Docker Inspector with an image reference versus an image that is saved to a .tar file, it uses the *docker-java* library to access the Docker registry to pull the image.

If *docker pull {targetimage]* works from the command line, then Detect Docker Inspector is also able to pull that image, because docker-java can be configured the same way as the Docker command line utility.

There are other ways to configure docker-java. For more information on configuring docker-java and Detect Docker Inspector for your Docker registry, refer to the configuration information in *docker-java documentation*.

Detect Docker Inspector does not override any of the configuration settings in the code, so any of the other methods (properties, system properties, system environment) work.

If you choose to use environment variables, and you are calling Detect Docker Inspector from Synopsys Detect, you must prefix the environment variable names with *DETECT_DOCKER_PASSTHROUGH_* to instruct Synopsys Detect to pass them on to Detect Docker Inspector. In that scenario, instead of *export SOMENAME=value*, use *export DETECT_DOCKER_PASSTHROUGH_SOMENAME=value*.

If you choose to use system properties which are normally set using *java -D*, and you are calling Detect Docker Inspector from Synopsys Detect, you must put the properties in a file; for example, *mydockerproperties.properties*, and use

```
--detect.docker.passthrough.system.properties.path=mydockerproperties.p
roperties
```

to point Detect Docker Inspector to those property settings.

Troubleshooting overview

To troubleshoot issues with Detect Docker Inspector, run with DEBUG logging enabled:

```
--logging.level.com.synopsys=DEBUG
```

Considerations when running on Windows

Docker file sharing settings

Detect Docker Inspector requires the ability to share directories with the image inspector containers. It shares a directory with image inspector containers by mounting it as a volume. Configure your Docker settings to enable this file sharing by adding your home directory as a sharable directory on the Docker settings Resources > FILE SHARING screen.

The shared directories are created under the Synopsys Detect output directory (controlled by Synopsys Detect *detect.output.path*). If you change the location of the Synopsys Detect output directory be sure your Docker file sharing settings enable sharing of that directory.

Docker restrictions

Docker on Windows has restrictions that impact Detect Docker Inspector:

1. Docker can be configured to pull either Linux images, or Windows images. You can see how your Docker installation is configured by looking at the *OSType* value in the output of the *docker info* command. If Docker is configured for Linux images, it cannot pull Windows images, and vice versa. The command to change Docker's *OSType* value appears in the Docker Desktop menu. Refer to Docker documentation for more information.
2. When pulling Windows images, Docker requires that the architecture of the pulled image matches the architecture of your machine, and that the Windows version of the pulled image is a close match to the Windows version of your machine.

Problems and solutions

The following suggestions are related to specific problems.

Problem: When directly invoking the .jar file, an error message displays "Malformed input or input contains unmappable characters."

Possible cause: Your local character encoding does not match the target container file system character encoding.

Solution/workaround: Set the character encoding to UTF-8 when invoking Java:

```
java -Dfile.encoding=UTF-8 ...
```

Problem: Property values are set in unexpected ways.

Possible cause: Detect Docker Inspector is built using the Spring Boot application framework. Spring Boot provides a variety of ways to set property values. This can produce unexpected results if, for example, you have an environment variable whose name maps to a Detect Docker Inspector property name. Refer to the *Spring Boot documentation on external configuration* for more details.

Problem: The image inspector service cannot write to the mounted volume; SELinux is enabled.

When this happens, the following error may appear in the container log:

```
Exception thrown while getting image packages: Error inspecting image:
[container_image_inspector_dir_path]/shared/run_.../output/..._containe
rfilesystem.tar.gz (Permission denied)
...
Caused by: java.io.FileNotFoundException: /opt/blackduck/hub-imageinspe
ctor-ws/shared/run_.../output/..._containerfilesystem.tar.gz (Permissio
n denied)
```

Possible cause: SELinux policy configuration.

Solution/workaround: Add the *svirt_sandbox_file_t* label to Detect Docker Inspector's shared directory. This enables the Detect Docker Inspector services running in Docker containers to write to it:

```
sudo chcon -Rt svirt_sandbox_file_t /tmp/[project_name]-files/shared/
```

Problem: The image inspector service cannot read from the mounted volume.

When this happens, the following error may appear in the container log:

```
Error inspecting image: [container_image_inspector_dir_path]/shared/run
_.../{image}.tar (Permission denied)
```

Possible cause: The Linux umask value on the machine running Detect Docker Inspector is too restrictive.

Solution/workaround: Set the umask value to 022 when running Detect Docker Inspector. The cause could be a umask value that prevents read access to the file, or read or execute access to the directory. Detect Docker Inspector requires a umask value that does not remove read permissions from files and does not remove read or execute permissions from directories. For example, a umask value of 022 works.

Problem: Detect Docker Inspector cannot perform Docker operations because the remote access port is not enabled on the Docker engine.

When this happens, the following error may appear in the log:

```
Error inspecting image: java.io.IOException: Couldn't load native libra
ry
Stack trace: javax.ws.rs.ProcessingException: java.io.IOException: Coul
dn't load native library
```

In versions of Detect Docker Inspector prior to 8.2.0, the logged error was:

```
Error inspecting image: Could not initialize class org.newsclub.net.uni
x.NativeUnixSocket
Stack trace: java.lang.NoClassDefFoundError: Could not initialize class
 org.newsclub.net.unix.NativeUnixSocket
```

Possible cause: The TCP socket for remote access is not enabled on the Docker engine. For more information, refer to *Docker documentation*.

Solution/workaround: Follow the instructions in the *Docker documentation* to open the TCP port on the Docker engine.

Detect Docker Inspector Release notes

Version 10.0.1

Dependency update

- Updated internal build dependencies for Image inspector library to 14.1.4, Integration rest library to 10.3.6 and Image Inspector Web Service to 5.0.15

Version 10.0.0

Changed features

- Changed name from Black Duck Docker Inspector to Detect Docker Inspector.
- Changed the default value for imageinspector.service.log.length from 10,000 lines to 50,000 lines.

Removed features

- Removed support for running Docker Inspector as a standalone utility (by executing blackduck-docker-inspector.sh or by executing the Detect Docker Inspector .jar). Detect Docker Inspector must be invoked by running Synopsys Detect.
- Removed all Detect Docker Inspector properties involved in connecting to Black Duck ("blackduck.*"). Use Synopsys Detect properties instead.

Version 9.4.3

Dependency update

- Upgraded to Spring Boot version 2.6.6 / Spring version 5.3.18.

Version 9.4.2

Resolved issue

- (IDOCKER-764) Improved the clarity of the error message returned when the file provided via the *docker.tar* property does not have the supported (UNIX tar) format.

Version 9.4.1

Documentation updates

- Added to the documentation the *organize components by layer* feature (properties bdio.organize.components.by.layer and bdio.include.removed.components).

Version 9.4.0

New features

- Enhanced Black Duck Docker Inspector to provide information on relationships between linux package manager components (ie. discern direct from transitive dependencies).

Changed features

- Deprecated properties associated with connection to Black Duck. This is a consequence of the transition away from supporting the use of Black Duck Docker Inspector as a standalone utility (as supposed to scanning via Detect).

Version 9.3.1

Resolved issue

- (IDETECT-2942) Resolved issue that caused Black Duck Docker Inspector to try to read the body of a bad response from image inspector services without checking the response's status code to make sure it was a successful request. Black Duck Docker Inspector now throws an exception if it receives a bad response.

Version 9.3.0

New feature

- Added support for Open Container Initiative (OCI) images provided to Black Duck Docker Inspector using the *docker.tar* property.

Resolved issue

- (IDOCKER-742) Resolved an issue that caused Black Duck Docker Inspector to fail to find the target image (requested using the *docker.image.repo* property) in a multi-image .tar file when the *docker.image.repo* value includes the registry prefix (e.g. "docker.io/").

Version 9.2.3

Resolved issue

- (IDOCKER-736) Resolved an issue that could cause Black Duck Docker Inspector to use the wrong component namespace in BDIO, resulting in an empty BOM, for Oracle Linux images.

Version 9.2.2

Resolved issue

- (IDOCKER-722) Resolved an issue that caused Black Duck Docker Inspector to, when it found a "white out opaque directory" file, to delete files added by the current layer as supposed to only deleting files added by lower layers.

Version 9.2.1

Resolved issue

- (IDOCKER-727) Resolved an issue that caused Black Duck Docker Inspector to fail to discover packages in CentOS 7 based images due to an error upgrading the rpm database.

Version 9.2.0

New features

- Added the ability to pull and inspect a specified platform of a multi-platform image using the new docker.image.platform property.

Changed features

- Added support for running Black Duck Docker Inspector using Java 15

Resolved issue

- (IDOCKER-715) Resolved an issue that could cause Black Duck Docker Inspector to fail on Windows during a docker pull operation with the message "java.lang.NoSuchMethodError: com.sun.jna.Native.load(Ljava/lang/String;Ljava/lang/Class;Ljava/util/Map;)Lcom/sun/jna/Library;".
- (IDOCKER-716) Resolved an issue that caused Black Duck Docker Inspector to discover no packages on fedora:33 and fedora:34 based images.

Version 9.1.1

Resolved issue

- (IDOCKER-710) Resolved an issue that could cause Black Duck Docker Inspector to fail when the target image file system contains circular symbolic links.

Version 9.1.0

New features

- Added support for running on Windows 10 Enterprise Edition by executing the Black Duck Docker Inspector .jar file directly.
- Added property use.platform.default.docker.host (default to true).
- Added property imageinspector.service.log.length (defaults to 10000 lines). This gives the user control over the number of lines of the imageinspector service log that are included in the Detect log when logging level is set to DEBUG or higher.

Changed feature

- Changed default working directory from /tmp to $HOME/blackduck/docker-inspector

Resolved issue

- (IDOCKER-709) Resolved an issue that could cause Black Duck Docker Inspector to fail because it ran out of memory while writing the image inspector log to the Black Duck Docker Inspector log.

Version 9.0.2

Resolved issues

- (IDOCKER-706) Resolved an issue that could cause Black Duck Docker Inspector to fail when using existing image inspector services when given a target docker .tar file that resided outside the directory shared with the image inspector container(s).

Version 9.0.1

Resolved issues

- Resolved an issue that could cause files to be omitted from the squashed image produced by Black Duck Docker Inspector. The problem occurred on images that declared a directory opaque and added files to that directory within the same layer that declared it opaque.

Version 9.0.0

Changed feature

- The internal format of the Black Duck Input Output (BDIO) file that is produced is now compatible with Synopsys Detect version 6.3 and later.

Version 8.3.1

Resolved issues

- Fixed an issue that prevented the *linux.distro* property from working correctly.

Version 8.3.0

New features

- Docker Inspector now writes a summary of results to the file results.json, located in the output directory.

Version 8.2.3

Resolved issues

- Eliminated the Spring banner from the log to facilitate piping help output through a Markdown formatter.

Version 8.2.2

Resolved issues

- Increased the default value of *service.timeout* from four minutes to ten minutes.
- The time Docker Inspector waits for an image inspector service to come online is now controlled using the *service.timeout* property.

Version 8.2.1

Resolved issues

- Resolved an issue that caused *blackduck-docker-inspector.sh* to display the help overview even when a different help topic is requested.
- Resolved an issue that caused Docker Inspector to return the full container filesystem even when only application components are requested (*docker.platform.top.layer.id* is specified).

Version 8.2.0

New features

- Added support for Java 11.
- Added the ability to generate help by topic (--help {topic}).
- Added the ability to generate help in HTML.
- Added the ability to write help output to a given file.

Version 8.1.6

Changed feature

- Adjusted logging to ensure that sensitive information does not display in a debug log.

Version 8.1.5

Resolved issues

- Resolved an issue that could cause Docker Inspector to incorrectly identify the package manager of the target image.

Version 8.1.4

New features

- Added a GitLab continuous integration deployment example.

Version 8.1.3

New features

- Added a Travis continuous integration deployment example.

Version 8.1.2

Changed features

- Updated *blackduck-docker-inspector.sh* to download the Docker Inspector .jar file from the new Artifactory repository at sig-repo.synopsys.com.

Version 8.1.1

Resolved issues

- Resolved an issue that caused Docker Inspector to fail when it was unable to read the image inspector container log.

Version 8.1.0

Resolved issues

- Resolved an issue that could cause a *No such file* error on files named *classes.jsa* when inspecting images containing Java.

New features

- Added the property *output.containerfilesystem.excluded.paths*.
- Added the command line switch *--help=true* for invoking help.
- Added the property *output.include.squashedimage*.

Version 8.0.2

Resolved issues

- Resolved an issue that could cause missing components on Fedora-based Docker images.

Version 8.0.1

Resolved issues

- Resolved an issue that could cause OpenSUSE components to be omitted from the Bill Of Materials.

New features

- Increased default image Inspector service timeout from two minutes to four minutes.

Version 8.0.0

New features

- Added the ability to collect only those components added to the image package manager database by your application layers, versus the platform layers on which your application is built.
- Added the ability to provide the full code location name.

Removed features

- Docker exec mode (deprecated in Docker Inspector 7.0.0) is removed. Docker Inspector now supports HTTP client mode only.

Version 7.3.3

Resolved issues

- Resolved an issue that could prevent controlling the image inspector HTTP request service timeout through the property *service.timeout*.

Version 7.3.2

Resolved issues

- Resolved an issue that could cause RPM packages containing a value in the epoch field to be missing from the Black Duck Bill of Materials (BOM).

Version 7.3.1

Resolved issues

- Resolved an issue that could cause Hub Detect versions 5.2.0 and higher to fail with an error message of *DOCKER extraction failed: null* when invoking Docker Inspector on a non-Linux Docker image.

Version 7.3.0

New features

- Added the property *system.properties.path*.

Version 7.2.4

Resolved issues

- Resolved an issue that could cause Docker Inspector to fail with an error message of *Error inspecting image: Failed to parse docker configuration file (Unrecognized field "identitytoken")*.

Version 7.2.3

Changed features

- When constructing the container file system with the logging level set to DEBUG or TRACE: after applying each image layer, Docker Inspector now logs contents of the layer's metadata (json) file and the list of components.

Version 7.2.2

Resolved issues

- Resolved an issue that could cause Black Duck input/output data (BDIO) uploads to fail for openSUSE and Red Hat Docker images.

Version 7.2.1

Resolved issues

- Resolved an issue that could cause the Black Duck BOM creation to fail with an error message of *Error in MAPPING_COMPONENTS* displayed on the Black Duck Scans page for certain images.

Version 7.2.0

New features

- Added offline mode.

Resolved issues

- Resolved an issue that could generate a warning message of *Error creating hard link* to be logged when inspecting certain images.
- Resolved an issue that could generate a warning message of *Error removing whited-out file .../.wh..opq* to be logged when inspecting images with opaque directory whiteout files.
- Reduced the disk space used in the working directory within the Inspector containers.

Version 7.1.0

Changed features

- Modified the format of the generated external identifiers to take advantage of the Black Duck KnowledgeBase preferred alias namespace feature.
- Modified the format of the generated external identifiers to include the epoch, when applicable, for RPM packages.

Version 7.0.1

Resolved issues

- When the logging level is set to DEBUG or higher, the contents of the image inspector service log are now included in the log output.
- The image inspector service is now started with the same logging level as Black Duck Docker Inspector.

Version 7.0.0

Changed features

- Hub Docker Inspector is now renamed to Black Duck Docker Inspector. The shell script, .jar filename, properties, and code blocks are updated accordingly.
- Black Duck Docker Inspector now runs in HTTP mode by default.

Resolved issues

- HTTP client mode: Resolved an issue that prevents removal of the image inspector image upon completion.
- HTTP client mode: Resolved an issue with user-specified Black Duck project names and version names.
- HTTP client mode: Resolved an issue that caused the container filesystem to be provided even when it was not requested.

Version 6.3.1

Resolved issues

- Resolved an issue that could cause DEBUG-level warnings to be logged while inspecting images with file paths containing non-ASCII characters.
- Improved logging for when the image inspector service is started but never comes online.
- In http client mode > start service mode: if a health check fails, Docker Inspector now performs a *docker logs* operation on the container to reveal the root problem.
- Orchestration platform properties are now included in the *--help* output.

Version 6.3.0

New features

- Added the *--inspectorimagefamily* command line argument, which prints the Inspector image family name.

Version 6.2.0

Resolved issues

- Resolved an issue that caused Docker Inspector to fail when the image repository contained a : character and the image tag was not specified.

New features

- Added the *--pullairgapzip* option.
- Improved error messages.

Version 6.1.0

Resolved issues

- Added support for running on container application platforms such as Kubernetes and OpenShift.
- Renamed Rest Client Mode to HTTP Client Mode.
- Resolved an issue that prevented BDIO (Black Duck Input/Output data) from being uploaded to the Hub. This issue only impacted HTTP client mode.
- Resolved an issue that caused Docker Inspector to fail when inspecting an image containing no package manager. This issue only impacted HTTP client node.

Version 6.0.4

Resolved issues

- Resolved an issue wherein Hub Docker Inspector may fail if the target Docker tarfile path contained spaces.

Version 6.0.3

Resolved issues

- Resolved an issue causing Hub Docker Inspector to produce an unnecessarily large container filesystem output file.

Version 6.0.2

Resolved issues

- Resolved an issue causing Hub Docker Inspector to fail when the image exists in the local cache but not the registry.

Version 6.0.1

Resolved issues

- Removed extraneous and possibly misleading log messages.

New features

- Added the properties *docker.image.repo* and *docker.image.tag* to the usage message generated when using the command line argument *--help*.

Version 6.0.0

New features

- Added REST client mode.

Changed features

- The available properties list included in the usage message, which displays when using the command line argument *--help*, is now sorted alphabetically.

- The format of the (optional) container filesystem output file name has changed. The new container system file name is *{image name}{image tag} containerfilesystem.tar.gz* or *{image tarfilename}.tar.gz*, depending on how the target image is specified.

Git project support

Unlike most detectors, the Git detectors do not discover dependencies; they only discover project information. Regardless of which Git detector runs, it discovers as many of the following as it can find: project name, project version (branch), git repository URL, and commit hash.

Ideally a Git detector runs in combination with a package manager detector that discovers dependencies. If the package manager detector is unable to derive project and project version names, the Git detector may be able to provide them.

A Git detector will run if Synopsys Detect finds a .git subdirectory in your source directory.

If Synopsys Detect finds a git executable (see the *detect git executable* property) the Git CLI detector will run git commands and derive project information from the output.

Otherwise the Git Parse detector will attempt to parse (and derive project information from) *config*, *HEAD*, and *ORIGIN_HEAD* files within the .git subdirectory. The Git Parse detector will only be able to discover (and supply to Black Duck) the git commit hash if it finds an *ORIGIN_HEAD* file in the .git directory.

GoLang support

Related properties

Detector properties

Overview

Synopsys Detect has four detectors for GoLang:

- Go Mod Cli (GO_MOD) detector (recommended)
- Go Lock (GO_DEP) detector
- Go Gradle (GO_GRADLE) detector
- Go Vendor (GO_VENDOR) detector
- Go Vndr (GO_VNDR) detector

Go Mod Cli (GO_MOD) detector

- Discovers dependencies of go language (GoLang) projects.
- Attempts to run on your project if a go.mod file is found in your source directory.
- Requires the *go* executable to be on the PATH or the executable path to be set with *detect.go.path*.
- Runs *go list -m* and *go mod graph*, and parses the output of both to discover dependencies.
- Runs *go mod why* to remove unused Go modules such as dependencies required by the build system or tests.

Excluding Test and Build System dependencies

Synopsys Detect can run additional Go commands to filter out *test* and *build system* dependencies from the BOM.

Use *detect.go.mod.dependency.types.excluded=VENDORED* to exclude the most dependencies. This will instruct Synopsys Detect to execute `go mod why -vendor` to generate a list of modules to exclude.

Use the VENDORED option because running `go mod why` without the `-vendor` flag results in *test* and *build system* dependencies being included in the BOM from Go modules declaring a version prior to `Go 1.16`. See the *go mod why documentation* for additional details.

Note on current exclusion behavior:

Now, Synopsys Detect runs *go mod why* by default to remove unused components from the BOM. This may result in a low number of detected dependencies. This behavior can be controlled with the *detect.go.mod.dependency.types.excluded* property.

If the *detect.go.mod.dependency.types.excluded* property is not provided, the behavior is driven by the value of this deprecated property *detect.go.mod.dependency.types*.

In version 8.0.0, Synopsys Detect will not exclude any dependencies from the BOM by default and *detect.go.mod.dependency.types* will be removed.

Go Lock (GO_DEP) detector

- Discovers dependencies of GoLang projects.
- Attempts to run on your project if a Gopkg.lock file is found in your source directory.
- Does not rely on external executables; for example, go, dep, and others.
- Parses Gopkg.lock for dependencies.

Go Gradle (GO_GRADLE) detector

- Discovers dependencies of go language (GoLang) projects.
- Attempts to run on your project if a gogradle.lock file is found in your source directory.
- Does not rely on external executables; for example, go, dep, and others.
- Parses gogradle.lock for dependencies.

Go Vendor (GO_VENDOR) detector

- Discovers dependencies of go language (GoLang) projects.
- Attempts to run on your project if the file vendor/vendor.json is found in your source directory.
- Does not rely on external executables; for example, go, dep, and others.
- Parses vendor/vendor.json for dependencies.

Go Vndr (GO_VNDR) detector

- Discovers dependencies of go language (GoLang) projects.
- Attempts to run on your project if the file vendor.conf is found in your source directory.
- Does not rely on external executables; for example, go, dep, and others.
- Parses vendor.conf for dependencies.

Gradle support

Related properties

Detector properties

Note: Gradle Project Inspector relies on the Project Inspector tool thus does not accept Gradle specific configuration properties.

Overview

Synopsys Detect has two detectors for Gradle:

- Gradle Native Inspector
- Gradle Project Inspector

Gradle Native Inspector

- Discovers dependencies of Gradle projects.
- Will run on your project if it finds a build.gradle file in the top level source directory.

Gradle Native Inspector requires either gradlew or gradle:

1. Synopsys Detect looks for gradlew in the source directory (top level). You can override this by setting the Gradle path property. If not overridden and not found:
2. Synopsys Detect looks for gradle on $PATH.

Runs `gradlew gatherDependencies` to get a list of the project's dependencies, and then parses the output.

Gradle Native Inspector allows you to filter projects based on both the name and the path. The path is unique for each project in the hierarchy and follows the form ":parent:child". Both filtering mechanism support wildcards.

The inspector defines the custom task 'gatherDependencies' with the help of a Gradle script (`init-detect.gradle`), which it usually downloads automatically. The file init-detect.gradle has a dependencies on ExcludedIncludedFilter, ExcludedIncludedWildcardFilter, and IntegrationEscapeUtil that come from https://github.com/blackducksoftware/integration-common. Filtering (including/excluding projects and configurations) is performed by the Gradle/Groovy code to control the output of the `dependencies` Gradle task invoked by the 'gradlew gatherDependencies' command.

The init-detect.gradle script configures each project with the custom 'gatherDependencies' task, which invokes the 'dependencies' Gradle task on each project. This ensures the same output is produced as previous versions. The inspector consumes the output of `gradlew gatherDependencies` task.

Running the Gradle inspector with a proxy

Synopsys Detect will pass along supplied *proxy host* and *proxy port* properties to the Gradle daemon if applicable.

Gradle detector buildless

The buildless gradle detector uses Project Inspector to find dependencies and does not support dependency exclusions.

It currently supports "build.gradle" and does not support Kotlin build files.

Erlang/Hex/Rebar support

Related properties

Detector properties

Overview

The Rebar detector discovers dependencies of Erlang projects that use the Hex package manager.

The Rebar detector runs if Synopsys Detect finds a *rebar.config* file in your project. A *rebar3* executable must be found on the PATH, or must be *provided*.

The Rebar detector runs the *rebar3 tree* command and parses the output for dependency information.

Ivy (Ant) support

Overview

Synopsys Detect supports an Ivy detector to extract dependency information for projects that use Ivy, a common dependency manager for Ant projects.

Synopsys Detect runs the Ivy detector if it finds a ivy.xml file in your project.

The Ivy detector parses the ivy.xml file for information on your project's dependencies. The Ivy detector extracts the project's name and version from the build.xml file. If it does not find a build.xml file, it will defer to values derived by git, from the project's directory, or defaults.

Lerna support

Related properties

Detector properties

Overview

The Lerna detector will register in the presence of a lerna.json file.

It will then execute a lerna command to retrieve all the packages defined in the project.

Each package has a location within the project structure.

It is expected to find a package.json and some type of lock file. Supported lockfile types are package-lock.json, npm-shrinkwrap.json, and yarn.lock.

If no lockfile is present in the package, it will be assumed that all the dependencies defined within the package's package.json file will be resolved in the lockfile at the root of the project. If no lockfile is present at the root of the project, Lerna extraction will fail.

Extracting from package-lock.json

The Lerna detector will execute the same code as the *NPM package lock detector*.

The *NPM package lock detector* related properties also apply.

Since the Lerna detector is currently not using the NPM Cli, the only property that applies is *detect.npm.dependency.types.excluded*.

Extracting from npm-shrinkwrap.json

The Lerna detector will execute the same code as the *NPM shrinkwrap detector*.

The *NPM shrinkwrap detector* related properties also apply.

Since the Lerna detector is currently not using the NPM Cli, the only property that applies is *detect.npm.dependency.types.excluded*.

Extracting from yarn.lock

The Lerna detector will execute the same code as the *Yarn detector*.

The *Yarn detector related properties* also apply.

Yarn workspaces are not currently supported by the Lerna detector.

Private packages

With the *detect.lerna.package.types.excluded* property, users can specify whether or not to include private packages as defined by Lerna.

Lerna path

Synopsys Detect executes commands against the Lerna executable to determine package information.

Synopsys Detect will attempt to find the Lerna executable, but if the user wishes to override the executable Synopsys Detect uses, they can supply a path to the executable using *detect.lerna.path*

Excluding Packages

The Lerna detector includes/excludes Lerna packages found when it runs `lerna ls --all --json` as specified by *detect.lerna.packages.included* and *detect.lerna.packages.excluded*.

Maven support

Related properties

Detector properties

Overview

Synopsys Detect has three detectors for Maven:

- Maven CLI
- Maven Wrapper CLI
- Maven Project Inspector

Note:

- Maven Project Inspector relies on Project Inspector thus does not accept Maven specific configuration properties.

Maven CLI

- Discovers dependencies of Maven projects by executing mvn commands.

- Will run on your project if it finds a pom.xml file in the top level source directory and requires either `mvnw` or `mvn`.

1. Synopsys Detect looks for `mvnw` in the source directory (top level). You can override this by setting the Maven path property.
2. If `mvnw` path is not overridden and `mvnw` is not found:Synopsys Detect looks for mvn on $PATH.

The Maven CLI detector runs `mvn dependency:tree` to get a list of the project's dependencies and then parses the output. Synopsys Detect assumes the output of this command will be in Maven's default logging format. Customizations of Maven's logging format can break Synopsys Detect's parsing.

Scope inclusion/exclusion is performed during the parsing of the output of the `mvn dependency:tree` command.

Components with unknown graph locations

The output of `mvn dependency:tree` does not always provide information on a component's positions in the dependency graph. When this information is missing, Synopsys Detect will place the component under a placeholder "component" named *Additional_Components*.

For example: Imagine a project with two scopes (compile and test), two direct dependencies A and B, both of which depend transitively on C, and imagine we want Synopsys Detect to exclude test scope from its output. The actual dependency graphs for this project look like:

```
compile scope:
A
\- C

test scope:
B
\- C
```

For this project, the output of `mvn dependency:tree` may only show:

```
compile scope:
A

test scope:
B
\- C (compile)
```

From that output we can tell that C is part of the compile scope, but there is no information about where in the compile scope graph C belongs. Its position in the test scope is irrelevant since test scope is being excluded. Rather than excluding C in this case, Synopsys Detect puts it under the placeholder "component" named *Additional_Components*.

Maven Wrapper CLI

The Maven Wrapper CLI detector attempts to run on your project if it finds a pom.groovy file in the source directory (top level), and then operates exactly as the Maven CLI detector does.

Maven Project Inspector

The Maven Project Inspector detector uses Project Inspector, which currently does not support plugins.

NPM support

Related properties

Detector properties

Overview

Synopsys Detect has the following NPM detectors:

- NPM package shrinkwrap detector
- NPM package lock detector
- NPM Package Json Parse detector
- NPM CLI detector

Excluding dependency types

Synopsys Detect offers the ability to exclude package manager specific dependency types from the BOM. NPM dependency types can be filtered with the *detect.npm.dependency.types.excluded* property.

NPM shrinkwrap

The NPM shrinkwrap detector uses the *npm-shrinkwrap.json* file.

It is the same as a package-lock.json but is a publishable lockfile. Though both have high accuracy, Synopsys Detect prefers shrinkwrap over package lock.

NPM package lock

The NPM package lock runs when it finds a *package-lock.json* generated by NPM.

For accurate filtering with the *detect.npm.dependency.types.excluded* property, a package.json file is also required.

NPM automatically generates the package-lock.json after an *npm install* which should be run prior to scanning.

NPM CLI detector

The NPM CLI detector requires a package.json and an npm executable.

It executes *npm ls -json*, and verifies that it succeeded by ensuring that no messages were written to stderr, and that the exit code was 0. If both conditions are met, it parses the output for dependencies.

If *npm ls -json* does write messages to stderr (for example, lint messages) that you would like Synopsys Detect to ignore, you can silence errors by adding the --silent npm argument to the npm command using Synopsys Detect property --detect.npm.arguments.

This detector is only invoked if the more accurateNPM detectors (above) cannot be run.

NPM Package Json Parse

This is the least accurate of the NPM detectors.

It requires and parses a package.json file.

The NPM Package Json Parse detectable is unable to discover transitive dependencies.

When generating BDIO, the NPM Package Json Parse detectable uses version strings exactly as it finds them in package.json. Components whose versions are specified in the package.json using wildcards ("^", etc.) will not match components in the KB due to the wildcards.

NuGet support

Related properties

Detector properties

Overview

The NuGet detectors can discover dependencies of NuGet projects.

There are three NuGet detectors: the NuGet Solution Native Inspector, NuGet Project Native Inspector, and the NuGet Project Inspector. The detectors run a platform dependent self-contained executable that is currently supported on Windows, Linux and Mac platforms.

Note:

- NuGet Project Inspector relies on Project Inspector thus does not accept NuGet specific configuration properties.
- The NuGet Detectors do not work with mono.

Synopsys Detect NuGet Inspector

Source: https://github.com/blackducksoftware/detect-nuget-inspector

Binary: https://sig-repo.synopsys.com/artifactory/bds-integrations-release/com/synopsys/integration/detect-nuget-inspector/

Synopsys Detect NuGet Inspector on Alpine

The Synopsys Detect NuGet Inspectors depend on packages not installed by default on Alpine systems, such as the dynamic loader for DLLs.

When the dynamic loader is not present, an error message similar to the following appears in the log as a result of Synopsys Detect's attempt to execute the NuGet Inspector:

```
java.io.IOException: Cannot run program ".../tools/detect-nuget-inspect
or/detect-nuget-inspector-1.0.1-linux/detect-nuget-inspector" (in direc
tory ...): error=2, No such file or directory
```

To add these packages to an Alpine system:

```
apk add libstdc++ gcompat icu
```

Operation

An inspector is fully self-contained and requires no installation. Each executable is platform dependent and the correct inspector is downloaded by Synopsys Detect at runtime.

The NuGet Solution Native Inspector derives packages (dependencies) from solution (.sln) files.

The NuGet Project Native Inspector derives packages (dependencies) from project (.csproj, .fsproj, etc.) files. The supported project files are:

```
// C#
"*.csproj",
// F#
"*.fsproj",
// VB
"*.vbproj",
// Azure Stream Analytics
"*.asaproj",
// Docker Compose
"*.dcproj",
// Shared Projects
"*.shproj",
// Cloud Computing
"*.ccproj",
// Fabric Application
"*.sfproj",
// Node.js
"*.njsproj",
// VC++
"*.vcxproj",
// VC++
"*.vcproj",
// .NET Core
"*.xproj",
```

```
// Python
"*.pyproj",
// Hive
"*.hiveproj",
// Pig
"*.pigproj",
// JavaScript
"*.jsproj",
// U-SQL
"*.usqlproj",
// Deployment
"*.deployproj",
// Common Project System Files
"*.msbuildproj",
// SQL
"*.sqlproj",
// SQL Project Files
"*.dbproj",
// RStudio
"*.rproj"
```

The NuGet Solution Native Inspector runs if one or more solution (.sln) files are found.

The NuGet Project Native Inspector runs if no solution files are found, and one or more project files are found. Refer to the preceding list of project file types.

The NuGet inspectors derive dependency information from the first type of file in this order:

1. packages.config
2. project.lock.json
3. project.assets.json
4. project.json
5. XML of the project file

After discovering dependencies, NuGet client libraries are used to collect further information about the dependencies and write it to a JSON file (`<projectname>_inspection.json`). Synopsys Detect then parses that file for the dependency information.

NuGet detector buildless

In buildless mode, Synopsys Detect uses Project Inspector to find dependencies and only supports ".csproj" and ".sln" files.

pnpm support

Related properties

Detector properties

Overview

Synopsys Detect runs the pnpm detector if it finds a pnpm-lock.yaml file in your project, and parses the file to obtain information on your project's dependencies.

To specify which types of dependencies you want Synopsys Detect to exclude from the BOM (Dev and Optional dependencies) use the detect.pnpm.dependency.types.excluded property.

The pnpm detector extracts the project's name and version from the package.json file. If it does not find a package.json file, it will defer to a project name derived by git, from the project's directory, or defaults.

Python support

Related properties

Detector properties

Overview

Synopsys Detect has three detectors for Python:

- Pip detector
- Pipenv detector
- Pipfile lock detector
- Poetry detector

The Pip detector

The Pip detector discovers dependencies of Python projects.

The Pip detector attempts to run on your project if any of the following are true: a setup.py file is found, a requirements.txt is found, or a requirements file is provided using the *--detect.pip.requirements.path* property.

The Pip detector requires Python and pip executables:

- Synopsys Detect looks for python on $PATH. You can override this by setting *--detect.python.path*
- Synopsys Detect looks for pip on $PATH. You can override this by setting *--detect.pip.path*

The Pip detector runs the *pip-inspector.py script*, which uses Python/pip libraries to query the pip cache for the project, which may or may not be a virtual environment, for dependency information:

1. pip-inspector.py queries for the project dependencies by project name which can be discovered using setup.py, or provided using the detect.pip.project.name property, using the *pkg_resources library*. If your project is installed into the pip cache, this discovers dependencies specified in setup.py.
2. If one or more requirements files are found or provided, pip-inspector.py uses the Python API called parse_requirements to query each requirements file for possible additional dependencies, and uses the pkg_resources library to query for the details of each.

Ramifications of this approach:

- Because pip-inspector.py uses the pkg_resources library to discover dependencies, only those packages which have been installed; using, for example, `pip install`, into the pip cache; in other words, appearing in the output of `pip list`, are included in the output. There must be a match between the package version on which your project depends and the package version installed in the pip cache. Additional details are available in the *pkg_resources library documentation*.
- If the packages are installed into a virtual environment for your project, you must run Synopsys Detect from within that virtual environment.

Recommendations:

- Be sure that Synopsys Detect is finding the correct Python executable; this can be done by running the logging level at DEBUG and then reading the log. This is a particular concern if your system has multiple versions of Python installed; you must be sure Synopsys Detect is using the correct Python version.
- Create a setup.py file for your project.
- Install your project and dependencies into the pip cache:

```
python setup.py install
pip install -r requirements.txt
```

- The Pip detector derives your project name using your setup.py file if you have one. If you do not have a setup.py file, you must provide the correct project name using the propety --detect.pip.project.name.
- If there are any dependencies specified in requirements.txt that are not specified in setup.py, then provide the requirements.txt file using the Synopsys Detect property.
- If you are using a virtual environment, be sure to switch to that virtual environment when you run Synopsys Detect. This also applies when you are using a tool such as Poetry that sets up a Python virtual environment.

Pipenv detector

The Pipenv detector discovers dependencies of Python projects.

The Pipenv detector attempts to run on your project if either of the following is true:

1. A Pipfile is found.
2. A Pipfile.lock file is found.

The Pipenv detector also requires Python and Pipenv executables:

- Synopsys Detect looks for python (or python3 if the python3 property is set to true) on $PATH. You can override this by setting the python path property.
- Synopsys Detect looks for pipenv on $PATH.

The Pipenv detector runs `pipenv run pip freeze` and `pipenv graph --bare --json-tree` and derives dependency information from the output. The dependency hierarchy is derived from the output of `pipenv graph --bare --json-tree`. The output of `pipenv run pip freeze` is used to improve the accuracy of dependency versions.

To troubleshoot of the Pipenv detector, start by running `pipenv graph --bare --json-tree`, and making sure that the output looks correct since this is the basis from which Synopsys Detect constructs the BDIO. If the output of `pipenv graph --bare --json-tree` does not look correct, make sure the packages (dependencies) are installed into the Pipenv virtual environment (`pipenv install`).

Note: The *detect.pipfile.dependency.types.excluded* property does not apply to the Pipenv detector.

Pipfile lock detector

The Pipfile lock detector discovers dependencies of Python projects.

The Pipfile lock detector attempts to run on your project if either of the following is true AND neither of the Pip or Pipenv detectors apply:

1. A Pipfile.lock file is found.
2. A Pipfile file is found.

The Pipfile lock detector parses the Pipfile.lock file for dependency information. If the detector discovers a Pipfile file but not a Pipfile.lock file, it will prompt the user to generate a Pipfile.lock file by running `pipenv lock` and then run Synopsys Detect again. Pipfile.lock dependencies can be filtered using the *detect.pipfile.dependency.types.excluded* property.

Poetry detector

The Poetry detector discovers dependencies of Python projects.

The Poetry detector attempts to run on your project if either of the following is true:

1. A poetry.lock file is found.
2. A pyproject.toml file containing a tool.poetry section is found.

The Poetry detector parses poetry.lock for dependency information. If the detector discovers a pyproject.toml file but not a poetry.lock file, it will prompt the user to generate a poetry.lock by running `poetry install` and then run Synopsys Detect again. The Poetry detector extracts the project's name and version from the pyproject.toml file. If it does not find a pyproject.toml file, it will defer to values derived by git, from the project's directory, or defaults.

SBT support

Related properties

Detector properties

Overview

Synopsys Detect runs the dependecyDot task when it finds the following in your project

- build.sbt

The SBT detector requires a compatible dependency graph plugin and the "depedencyDot" task to generate dependency graphs for SBT projects. The plugin generates dot file graphs for each project and Synopsys Detect parses each dot graph into it's own code location.

Starting with SBT 1.4.0, a dependency graph plugin comes with SBT and can be enabled by adding `addDependencyTreePlugin` to "project/plugins.sbt".

For older versions of SBT, a *dependency graph plugin* can be installed *globally* or per project.

To install the plugin globally add the following to "$HOME/.sbt/1.0/plugins/build.sbt" where "1.0" is replaced by your SBT version.

```
addSbtPlugin("net.virtual-void" % "sbt-dependency-graph" % "0.10.0-RC1"
)
```

To install the plugin to a single project, add the same line to "src/project/plugins.sbt" where src is your project's source directory.

Synopsys Detect verfies the plugin is installed by running "sbt plugins" and looking for any of the following plugin names in the output. You can also perform this check yourself by running the same command.

```
sbt.plugins.DependencyTreePlugin
net.virtualvoid.sbt.graph.DependencyGraphPlugin
```

The plugins include a "dependencyDot" task which generates "target/configuration-dependencies.dot" for each project. In some cases, the dot files are not generated in "./target" and while those graphs will still be found and results are not affected, it could affect the project and version chosen and code location's directory.

**NOTE: Older SBT projects that generate a resolution cache are still supported but are being deprecated. You must install the plugin for SBT to continue working uninterrupted.

As of 8.8.0 of Synopsys Detect, evicted dependencies in Simple Build Tool(SBT) projects will not be included in the Bill of Materials(BoM) generated during the scan.

Background execution

The sbt command line utility is known to suspend when run in the background (this may be limited to Linux and Mac systems): *https://github.com/sbt/sbt/issues/701*. This can cause Synopsys Detect to suspend, if Synopsys Detect is run in the background, and the SBT detector runs. You can apply the workaround suggested in that github issue using the *detect.sbt.arguments* property:

```
./detect8.sh --detect.sbt.arguments="-Djline.terminal=jline.Unsupported
Terminal"
```

Swift & Xcode support

Related properties

Detector properties

Overview

Synopsys Detect has two detectors for Swift:

- Swift CLI detector
- Swift Package Resolved detector
- Xcode Project detector
- Xcode Workspace detector

All of these detectors primarily use the `Package.resolved` file to extract dependencies.

Swift CLI detector

The Swift CLI detector discovers dependencies of projects utilizing the Swift CLI.

The Swift CLI detector applies to directories containing a `Package.swift` file.

This detector requires a `swift` executable to run a command `swift package show-dependencies` to create a dependency graph of Swift packages.

All packages of the root node are considered direct dependencies in the BlackDuck BOM.

Packages found with a version of `unspecified` will appear in the BOM without a version.

Swift Package Resolved detector

The Swift Package Resolved detector discovers dependencies of projects utilizing the Swift Package Resolved.

The Swift Package Resolved detector applies to directories containing either a `Package.swift` file or `Package.resolved` file.

All packages of the packages are considered direct dependencies in the BlackDuck BOM.

Packages found with a version of `unspecified` will appear in the BOM without a version.

This detector does not require any executables to run.

Xcode Workspace detector

The Xcode Workspace detector discovers dependencies of Xcode projects utilizing *built-in tools within Xcode* for managing Swift dependencies.

The Xcode Workspace detector applies to directories matching the `*.xcworkspace` filename pattern within bounds of the *detect.detector.search.depth* property.

When Synopsys Detect finds a matching directory, the Xcode Workspace detector then searches for a `Package.resolved` file inside your `*.xcworkspace` directory at `*.xcworkspace/xcshareddata/swiftpm/Package.resolved` and extracts those dependencies.

Additionally, the Xcode Workspace detector will analyze the contents of the `*.xcworkspace/contents.xcworkspacedata` XML file to determine Xcode Workspace reference locations. This ignores the *detect.detector.search.depth* property as Synopsys Detect is no longer searching, but be directed to a specific location.

Today supported Workspace reference locations are directories and Xcode Projects. Any referenced locations that are not found currently trigger a failure of Synopsys Detect

- Xcode Projects are identified with by the suffix `*.xcodeproj`.
  - Example `location = "group:src/my-project.xcodeproj">`
- Directories are identified with by the suffix `/`
  - Example `location = "group:src/my-project/">`
  - Directories are assumed to be a built Swift Package containing a `Package.resolved` file at the specified location

This detector does not require any executables to run, but the Xcode Workspace must be buildable in Xcode.

Xcode Project detector

The Xcode Project detector discovers dependencies of Xcode projects utilizing *built-in tools within Xcode* for managing Swift dependencies.

The Xcode Project detector applies to directories matching the `*.xcodeproj` filename pattern within bounds of the *detect.detector.search.depth* property.

Once the Xcode Project detector finds a matching directory, it searches for a `Package.resolved` file inside your `.xcodeproj` directory at `[appName].xcodeproj/project.workspace/xcshareddata/swiftpm/Package.resolved`.

This detector does not require any executables to run, but the Xcode Workspace must be buildable in Xcode.

Package.resolved file

This file forms the basis for dependency extraction for most of the Swift and Xcode detectors.

The `Package.resolved` is a JSON file containing a flat list of Swift packages required by the project. This file can be empty if the project has no dependencies, in which case Synopsys Detect will create an empty code-location for the Xcode project.

Example `Package.resolved` file contents:

```
{
  "object": {
    "pins": [
      {
        "package": "swift-argument-parser",
        "repositoryURL": "https://github.com/apple/swift-argument-parse
r.git",
        "state": {
          "branch": null,
          "revision": "d2930e8fcf9c33162b9fcc1d522bc975e2d4179b",
          "version": "1.0.1"
        }
      }
    ]
  },
  "version": 1
}
```

Extraction method

Synopsys Detect is capable of extracting component name and versions from the JSON content within the `Package.resolved` file.

Forge

Currently, all packages are assumed to come from GitHub. For support of additional public repositories, please contact the Synopsys Detect support team.

Component Name

The component name is derived from the **repositoryURL** field.

```
"repositoryURL": "https://github.com/apple/swift-argument-parser.git"
```

Synopsys Detect will parse the path of the url to remove the host and `.git` extensions. In the above example, this produces a component name of `apple/swift-argument-parser`.

Component Version

Synopsys Detect will use the **state.version** field to identify the component version.

```
"state": {
    "branch": null,
    "revision": "d2930e8fcf9c33162b9fcc1d522bc975e2d4179b",
    "version": "1.0.1"
}
```

In the above example, this produces a component version of `1.0.1`.

Yarn support

Related properties

Detector properties

Overview

Synopsys Detect runs the Yarn detector if it finds both of the following files in your project:

- yarn.lock
- package.json

The Yarn detector reads both files (see *Yarn workspace support* below for information on other files the Yarn detector might read) to derive project and dependency information. The yarn.lock file must be up-to-date before Synopsys Detect runs. The package.json file specifies the direct dependencies for the project. Synopsys Detect adds these dependencies to the top level of the dependency graph

that it builds. The yarn.lock file contains necessary details about those direct dependencies and their transient dependencies, enabling Synopsys Detect to build the complete graph of direct and transient dependencies.

Synopsys Detect supports projects that use Yarn version 1 or version 2.

Yarn workspace support

In addition to the codelocation generated for the project (showing its direct and transitive dependencies), Synopsys Detect also generates a codelocation per included workspace (all workspaces are included by default).

Referencing workspaces

When you use the workspace exclude/include properties, refer to workspaces the same way Yarn refers to them in the *workspaces* list in the declaring package.json file: use the relative path of the workspace directory (relative to the declaring workspace).

For example, if your project package.json contains:

```
"workspaces": [
"packages/workspace-a",
"packages/workspace-b"
],
```

Synopsys Detect will expect you to refer to these workspaces as "packages/workspace-a" and "packages/workspace-b". This naming convention remains the same even at deeper workspace nesting levels. As an example, if your project has a workspace packages/workspace-a, and packages/workspace-a's package.json contains:

```
"workspaces": [
"child1-of-workspace-a",
"child2-of-workspace-a"
],
```

Synopsys Detect will expect you to refer to these workspaces as "child1-of-workspace-a" and "child2-of-workspace-a".

Excluding workspaces

By default, Synopsys Detect includes all workspaces in the results regardless of whether they are declared as dependencies of the project. You can specify a subset of workspaces to include or

exclude using the workspace exclude/include properties *detect.yarn.excluded.workspaces* and *detect.yarn.included.workspaces*.

When using the workspace exclude and include properties, use the workspace referencing guidelines described above. You can also use filename globbing-style wildcards and specify multiple values separated by commas.

11

Properties

See the left-hand menu for links to Synopsys Detect property settings.

Basic Properties

This page lists Synopsys Detect's basic properties. For advanced and deprecated properties, refer to *All Properties*.

bazel

Property	Description
detect.bazel.path	Bazel Executable: The path to the Bazel executable.
detect.bazel.target	Bazel Target: The Bazel target (for example, // foo:foolib) for which dependencies are collected. For Detect to run Bazel, this property must be set.
detect.bazel.cquery.options	Bazel cquery additional options: A comma-separated list of additional options to pass to the bazel cquery command.
detect.bazel.workspace.rules	default: NONE Acceptable Values: ALL, NONE, MAVEN_JAR, MAVEN_INSTALL, HASKELL_CABAL_LIBRARY, HTTP_ARCHIVE Bazel workspace rules: By default Detect discovers Bazel dependencies using all of the supported Bazel workspace rules that it finds in the WORKSPACE file. Alternatively you can use this property to specify the list of Bazel workspace rules Detect should use.

binary-scanner

Property	Description
detect.binary.scan.file.path	Binary Scan Target: If specified, this file and this file only will be uploaded for binary scan analysis. This property takes precedence over detect.binary.scan.file.name.patterns. The BINARY_SCAN tool does not provide project and version name defaults to Detect, so you need to set project and version names via properties when only the BINARY_SCAN tool is invoked.
detect.binary.scan.file.name.patterns	Binary Scan Filename Patterns: If specified, files in the source directory whose names match these file name patterns will be zipped and uploaded for binary scan analysis. This property will not be used if detect.binary.scan.file.path is specified. Search depth is controlled by property detect.binary.scan.search.depth. Directories specified via property detect.excluded.directories are excluded from the file search. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.
detect.binary.scan.search.depth	default: 0 Binary Scan Search Depth: When binary scan filename patterns are being used to search for binary files to scan, this property sets the depth at which Detect will search for files (that match those patterns) to upload for binary scan analysis.

bitbake

Property	Description
detect.bitbake.build.env.name	default: oe-init-build-env BitBake Init Script Name: The name of the build environment init script.
detect.bitbake.package.names	BitBake Package Names: A comma-separated list of package names from which dependencies are extracted.
detect.bitbake.source.arguments	BitBake Source Arguments: A comma-separated list of arguments to supply when sourcing the build environment init script.
detect.bitbake.search.depth	default: 1 BitBake Search Depth: The depth at which Detect will search for files generated by Bitbake.
detect.bitbake.dependency.types.excluded	default: NONE Acceptable Values: NONE, BUILD Bitbake Excluded Dependency Types: The dependency types to exclude from the results.

blackduck-server

Property	Description
blackduck.api.token	Black Duck API Token: The access token used to authenticate with the Black Duck Server.
blackduck.offline.mode	default: false Offline Mode: This can disable any Black Duck communication - if true, Detect will not upload BDIO files, it will not check policies, and it will not download and install the signature scanner.
blackduck.url	Black Duck URL: URL of the Black Duck server.

Property	Description
detect.test.connection	default: false Test Connection to Black Duck: Test the connection to Black Duck with the current configuration.
blackduck.offline.mode.force.bdio	default: false Force Offline BDIO Generation: This property will force Detect in offline mode to generate a BDIO even if no code locations were identified.

cleanup

Property	Description
detect.cleanup	default: true Cleanup Output: If true, the files created by Detect will be cleaned up.

conan

Property	Description
detect.conan.path	Conan Executable: The path to the conan executable.
detect.conan.dependency.types.excluded	default: NONE Acceptable Values: NONE, BUILD Conan Dependency Types Excluded: Set this value to indicate which Conan dependency types Detect should exclude from the BOM.
detect.conan.arguments	Additional Conan Arguments: A space-separated list of additional arguments to add to the 'conan info' command line when running Detect against a Conan project. Detect will execute the command 'conan info {additional arguments} .'

Property	Description
detect.conan.lockfile.path	Conan Lockfile: The path to the conan lockfile to apply when running 'conan info' to get the dependency graph. If set, Detect will execute the command 'conan info --lockfile {lockfile} .'
detect.conan.attempt.package.revision.match	default: false Attempt Package Revision Match: If package revisions are available (a Conan lock file is found or provided, and Conan's revisions feature is enabled), require that each dependency's package revision match the package revision of the component in the KB.

conda

Property	Description
detect.conda.environment.name	Anaconda Environment Name: The name of the anaconda environment used by your project.
detect.conda.path	Conda Executable: The path to the conda executable.

cpan

Property	Description
detect.cpan.path	cpan Executable: The path to the cpan executable.
detect.cpanm.path	cpanm Executable: The path to the cpanm executable.

dart

Property	Description
detect.dart.path	dart Executable: The path to the dart executable.
detect.flutter.path	flutter Executable: The path to the flutter executable.
detect.pub.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Dart Pub Dependency Types Excluded: Set this value to indicate which Dart pub dependency types Detect should exclude from the BOM.

debug

Property	Description
detect.diagnostic	default: false Diagnostic Mode: When enabled, diagnostic mode collects files valuable for troubleshooting (logs, BDIO file, extraction files, reports, etc.), writes them to a zip file, and logs the path to the zip file.

detector

Property	Description
detect.required.detector.types	Acceptable Values: BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE Required Detect Types: The set of required detectors.

docker

Property	Description
detect.docker.image	Docker Image Name: The Docker image name (repo:tag) to inspect.
detect.docker.image.id	Docker Image ID: The ID (shown in the 'IMAGE ID' column of 'docker images' output) of the target Docker image. The target image must already be local (must appear in the output of 'docker images').
detect.docker.path	Docker Executable: Path to the docker executable (used to load image inspector Docker images in order to run the Docker Inspector in air gap mode).
detect.docker.tar	Image Archive File: An image .tar file which is either a Docker image saved to a file using the 'docker save' command, or an Open Container Initiative (OCI) image .tar file. The file must be readable by all.

general

Property	Description
detect.target.type	default: SOURCE Acceptable Values: SOURCE, IMAGE Detect Target: Informs detect of what is being scanned which allows improved user experience when scanning different types of targets.
detect.wait.for.results	default: false Wait For Results: If set to true, Detect will wait for Synopsys products until results are available or the detect.timeout is exceeded.
detect.follow.symbolic.links	default: true Follow Symbolic Links: If set to true, Detect will follow symbolic links when searching for detectors, when searching for files that select detectors (such as Bitbake and Sbt) need, when searching for directories to exclude from signature scan, and when searching for binary scan targets. Symbolic links are not supported for Impact Analysis.

go

Property	Description
detect.go.path	Go Executable: Path to the Go executable.
detect.go.mod.dependency.types.excluded	default: NONE Acceptable Values: NONE, UNUSED, VENDORED Go Mod Dependency Types Excluded: Set this value to indicate which Go Mod dependency types Detect should exclude from the BOM.

gradle

Property	Description
detect.gradle.build.command	Gradle Build Command: Gradle command line arguments to add to the gradle/gradlew command line.
detect.gradle.configuration.types.excluded	default: NONE Acceptable Values: NONE, UNRESOLVED Gradle Configuration Types Excluded: Set this value to indicate which Gradle configuration types Detect should exclude from the BOM.
detect.gradle.path	Gradle Executable: The path to the Gradle executable (gradle or gradlew).

hex

Property	Description
detect.hex.rebar3.path	Rebar3 Executable: The path to the rebar3 executable.

iac-scan

Property	Description
detect.iac.scan.paths	IaC Scan Target Paths: A comma-separated list of paths to perform IaC scans on.
detect.iac.scan.arguments	IaC Scan Arguments: A space-separated list of additional arguments to use when running the IaC Scanner.
detect.iac.scanner.local.path	IaC Scanner Local Path: Use this property to specify the path to a local IaC Scanner.

impact-analysis

Property	Description
detect.impact.analysis.enabled	default: false Vulnerability Impact Analysis Enabled: If set to true, Detect will attempt to look for *.class files and generate a Vulnerability Impact Analysis Report for upload to Black Duck.
detect.impact.analysis.output.path	Impact Analysis Output Directory: The path to the output directory for Impact Analysis reports.

lerna

Property	Description
detect.lerna.path	Lerna Executable: Path of the lerna executable.
detect.lerna.package.types.excluded	default: NONE Acceptable Values: NONE, PRIVATE Lerna Package Types Excluded: Set this value to indicate which Lerna package types Detect should exclude from the BOM.

logging

Property	Description
logging.level.com.synopsys.integration	default: INFO Acceptable Values: OFF, ERROR, WARN, INFO, DEBUG, TRACE Logging Level: The logging level of Detect.

Property	Description
logging.level.detect	default: INFO Acceptable Values: OFF, ERROR, WARN, INFO, DEBUG, TRACE Logging Level Shorthand: Shorthand for the logging level of detect. Equivalent to setting *logging.level.com.synopsys.integration*.

maven

Property	Description
detect.maven.build.command	Maven Build Command: Maven command line arguments to add to the mvn/mvnw command line.
detect.maven.path	Maven Executable: The path to the Maven executable (mvn or mvnw).
detect.maven.included.scopes	Dependency Scope Included: A comma separated list of Maven scopes. Output will be limited to dependencies within these scopes (overridden by exclude).
detect.maven.excluded.scopes	Dependency Scope Excluded: A comma separated list of Maven scopes. Output will be limited to dependencies outside these scopes (overrides include).

npm

Property	Description
detect.npm.arguments	Additional NPM Command Arguments: A space-separated list of additional arguments that Detect will add at then end of the npm ls command line when Detect executes the NPM CLI Detector on an NPM project.

Property	Description
detect.npm.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV, PEER Npm Dependency Types Excluded: Set this value to indicate which Npm dependency types Detect should exclude from the BOM.
detect.npm.path	NPM Executable: The path to the Npm executable.

nuget

Property	Description
detect.nuget.config.path	Nuget Config File: The path to the Nuget.Config file to supply to the nuget exe.
detect.nuget.packages.repo.url	default: https://api.nuget.org/v3/index.json Nuget Packages Repository URL: The source for nuget packages

packagist

Property	Description
detect.packagist.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Packagist Dependency Types Excluded: Set this value to indicate which Packagist dependency types Detect should exclude from the BOM.

paths

Property	Description
detect.bash.path	Bash Executable: Path to the Bash executable.

Property	Description
detect.bdio.output.path	BDIO Output Directory: The path to the output directory for the generated BDIO file.
detect.bdio.file.name	BDIO File Name: The desired file name of BDIO file Detect produces in the BDIO Output Directory.
detect.detector.search.depth	default: 0 Detector Search Depth: Depth of subdirectories within the source directory to which Detect will search for files that indicate whether a detector applies.
detect.git.path	Git Executable: Path of the git executable
detect.java.path	Java Executable: Path to the Java executable used by Docker Inspector.
detect.output.path	Detect Output Path: The path to the output directory.
detect.scan.output.path	Scan Output Path: The output directory for all signature scanner output files. If not set, the signature scanner output files will be in a 'scan' subdirectory of the output directory.
detect.source.path	Source Path: The source path is the path to the project directory to inspect. If no value is provided, the source path defaults to the current working directory.
detect.tools.excluded	Acceptable Values: NONE, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN Detect Tools Excluded: The tools Detect should not allow, in a comma-separated list. Excluded tools will not be run even if all criteria for the tool is met. Exclusion rules always win.

Property	Description
detect.tools	Acceptable Values: ALL, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN Detect Tools Included: The tools Detect should allow in a comma-separated list. Tools in this list (as long as they are not also in the excluded list) will be allowed to run if all criteria of the tool are met. Exclusion rules always win.
detect.status.json.output.path	Status JSON Output Path: The directory to place a copy of the status.json file.
detect.scaaas.scan.path	SCAAAS Scan Target: This file will be uploaded to the BDBA worker for scan analysis in an SCA as a service environment.

pear

Property	Description
detect.pear.dependency.types.excluded	default: NONE Acceptable Values: NONE, OPTIONAL Pear Dependency Types Excluded: Set this value to indicate which Pear dependency types Detect should exclude from the BOM.
detect.pear.path	Pear Executable: The path to the pear executable.

pip

Property	Description
detect.pipfile.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Pipfile Dependency Types Excluded: A comma-separated list of dependency types that will be excluded.
detect.pip.project.name	PIP Project Name: The name of your PIP project, to be used if your project's name cannot be correctly inferred from its setup.py file.
detect.pip.project.version.name	PIP Project Version Name: The version of your PIP project, to be used if your project's version name cannot be correctly inferred from its setup.py file.
detect.pip.requirements.path	PIP Requirements Path: A comma-separated list of paths to requirements files, to be used to analyze requirements files with a filename other than requirements.txt or to specify which requirements files should be analyzed.
detect.pip.only.project.tree	default: false PIP Include Only Project Tree: By default, pipenv includes all dependencies found in the graph. Set to true to only include dependencies found underneath the dependency that matches the provided pip project and version name.
detect.pip.path	Pip Executable: The path to the Pip executable.
detect.pipenv.path	Pipenv Executable: The path to the Pipenv executable.

pnpm

Property	Description
detect.pnpm.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV, OPTIONAL pnpm Dependency Types: Set this value to indicate which pnpm dependency types Detect should exclude from the BOM.

project

Property	Description
detect.policy.check.fail.on.severities	default: NONE Acceptable Values: ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED Fail on Policy Violation Severities: A comma-separated list of policy violation severities that will fail Detect. If this is set to NONE, Detect will not fail due to policy violations. A value of ALL is equivalent to all of the other possible values except NONE.
detect.policy.check.fail.on.names	Fail on Policy Names with Violations: A comma-separated list of policy names with a non-zero number of violations that will fail Detect.
detect.project.description	Project Description: If project description is specified, your project will be created with this description. For updates, see detect.project.version.update.
detect.project.name	Project Name: An override for the name to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable project name. If that fails, the final part of the directory path where the inspection is taking place will be used.

Property	Description
detect.project.tier	Project Tier: If a Black Duck project tier is specified, your project will be created with this tier. For updates, see detect.project.version.update.
detect.project.version.name	Version Name: An override for the version to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable version name. If that fails, the current date will be used.
detect.project.version.nickname	Version Nickname: If a project version nickname is specified, your project version will be created with this nickname. For updates, see detect.project.version.update.
detect.project.version.notes	Version Notes: If project version notes are specified, your project version will be created with these notes. For updates, see detect.project.version.update.
detect.project.version.phase	default: DEVELOPMENT Acceptable Values: ARCHIVED, DEPRECATED, DEVELOPMENT, PLANNING, PRERELEASE, RELEASED Version Phase: If project version phase is specified, your project version will be created with this phase. For updates, see detect.project.version.update.
detect.project.version.update	default: false Update Project Version: If set to true, Detect will update the Black Duck project and project version according to configured project and project version properties. (By default, these properties are only set on created projects / project versions.)
detect.project.version.license	Project Version License: If project version license is specified, your project version will be created with this license. For updates, see detect.project.version.update.

python

Property	Description
detect.python.path	Python Executable: The path to the Python executable.

report

Property	Description
detect.notices.report	default: false Generate Notices Report: When set to true, a Black Duck notices report in text form will be created in your source directory.
detect.notices.report.path	Notices Report Path: The output directory for notices report. Default is the source directory.
detect.risk.report.pdf	default: false Generate Risk Report (PDF): When set to true, a Black Duck risk report in PDF form will be created.
detect.risk.report.pdf.path	Risk Report Output Path: The output directory for risk report in PDF. Default is the source directory.

ruby

Property	Description
detect.ruby.dependency.types.excluded	default: NONE Acceptable Values: NONE, RUNTIME, DEV Ruby Dependency Types Excluded: Set this value to indicate which Ruby(Gempsec) dependency types Detect should exclude from the BOM.

sbt

Property	Description
detect.sbt.path	Sbt Executable: Path to the Sbt executable.
detect.sbt.arguments	Additional sbt command Arguments: A space-separated list of additional arguments to add to sbt command line when running Detect against an SBT project. Detect will execute the command 'sbt {additional arguments} {Detect-added arguments}'.

signature-scanner

Property	Description
detect.blackduck.signature.scanner.arguments	Signature Scanner Arguments: A space-separated list of additional arguments to use when running the Black Duck signature scanner.
detect.blackduck.signature.scanner.copyright.search	default: false Signature Scanner Copyright Search: When set to true, user will be able to scan and discover copyright names in Black Duck. Corresponding Signature Scanner CLI Argument: --copyright-search.
detect.blackduck.signature.scanner.dry.run	default: false Signature Scanner Dry Run: If set to true, the signature scanner results are not uploaded to Black Duck, and the scanner results are written to disk via the Signature Scanner CLI argument: --dryRunWriteDir.

Property	Description
detect.blackduck.signature.scanner.individual.file.matching	default: NONE Acceptable Values: NONE, SOURCE, BINARY, ALL Individual File Matching: Users may set this property to indicate what types of files they want to match. Corresponding Signature Scanner CLI Argument: --individualFileMatching.
detect.blackduck.signature.scanner.license.search	default: false Signature Scanner License Search: When set to true, user will be able to scan and discover license names in Black Duck. Corresponding Signature Scanner CLI Argument: --license-search.
detect.blackduck.signature.scanner.local.path	Signature Scanner Local Path: To use a local signature scanner, specify the path where the signature scanner was unzipped. This will likely look similar to 'scan.cli-x.y.z' and includes the 'bin, icon, jre, and lib' directories of the expanded scan.cli.
detect.blackduck.signature.scanner.paths	Signature Scanner Target Paths: If this property is not set, the signature scanner target path is the source path (see property detect.source.path). If this property is set, the paths provided in this property's value will be signature scanned instead (the signature scanner will be executed once for each provided path).

Property	Description
detect.blackduck.signature.scanner.snippet.matching	default: NONE Acceptable Values: NONE, SNIPPET_MATCHING, SNIPPET_MATCHING_ONLY, FULL_SNIPPET_MATCHING, FULL_SNIPPET_MATCHING_ONLY Snippet Matching: Use this value to enable the various snippet scanning modes. For a full explanation, please refer to the 'Running a component scan using the Signature Scanner command line' section in your Black Duck server's online help. Corresponding Signature Scanner CLI Arguments: --snippet-matching, --snippet-matching-only, --full-snippet-scan.
detect.blackduck.signature.scanner.reduced.persistence	default: DEFAULT Acceptable Values: DEFAULT, RETAIN_UNMATCHED, DISCARD_UNMATCHED Reduced Persistence: Use this value to control how unmatched files from signature scans are stored. For a full explanation, please refer to *about reduced persistence signature scanning.*
detect.blackduck.signature.scanner.upload.source.mode	default: false Upload source mode: If set to true, the signature scanner will, if supported by your Black Duck version, upload source code to Black Duck. Corresponding Signature Scanner CLI Argument: --upload-source.
detect.excluded.directories.search.depth	default: 4 Detect Excluded Directories Search Depth: Enables you to adjust the depth to which Detect will search when creating signature scanner exclusion patterns.

swift

Property	Description
detect.swift.path	Swift Executable: Path of the swift executable.

yarn

Property	Description
detect.yarn.dependency.types.excluded	default: NONE Acceptable Values: NONE, NON_PRODUCTION Yarn Dependency Types Excluded: Set this value to indicate which Yarn dependency types Detect should exclude from the BOM.

All Properties

This page lists all Synopsys Detect properties, including advanced configuration and those properties that have been deprecated. For most use cases, refer to *basic properties*.

bazel

Property	Description
detect.bazel.path	Bazel Executable: The path to the Bazel executable.
detect.bazel.target	Bazel Target: The Bazel target (for example, // foo:foolib) for which dependencies are collected. For Detect to run Bazel, this property must be set.
detect.bazel.cquery.options	Bazel cquery additional options: A comma-separated list of additional options to pass to the bazel cquery command.

Property	Description
detect.bazel.workspace.rules	default: NONE Acceptable Values: ALL, NONE, MAVEN_JAR, MAVEN_INSTALL, HASKELL_CABAL_LIBRARY, HTTP_ARCHIVE Bazel workspace rules: By default Detect discovers Bazel dependencies using all of the supported Bazel workspace rules that it finds in the WORKSPACE file. Alternatively you can use this property to specify the list of Bazel workspace rules Detect should use.

binary-scanner

Property	Description
detect.binary.scan.file.path	Binary Scan Target: If specified, this file and this file only will be uploaded for binary scan analysis. This property takes precedence over detect.binary.scan.file.name.patterns. The BINARY_SCAN tool does not provide project and version name defaults to Detect, so you need to set project and version names via properties when only the BINARY_SCAN tool is invoked.
detect.binary.scan.file.name.patterns	Binary Scan Filename Patterns: If specified, files in the source directory whose names match these file name patterns will be zipped and uploaded for binary scan analysis. This property will not be used if detect.binary.scan.file.path is specified. Search depth is controlled by property detect.binary.scan.search.depth. Directories specified via property detect.excluded.directories are excluded from the file search. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Property	Description
detect.binary.scan.search.depth	default: 0 Binary Scan Search Depth: When binary scan filename patterns are being used to search for binary files to scan, this property sets the depth at which Detect will search for files (that match those patterns) to upload for binary scan analysis.

bitbake

Property	Description
detect.bitbake.build.env.name	default: oe-init-build-env BitBake Init Script Name: The name of the build environment init script.
detect.bitbake.package.names	BitBake Package Names: A comma-separated list of package names from which dependencies are extracted.
detect.bitbake.source.arguments	BitBake Source Arguments: A comma-separated list of arguments to supply when sourcing the build environment init script.
detect.bitbake.search.depth	default: 1 BitBake Search Depth: The depth at which Detect will search for files generated by Bitbake.
detect.bitbake.dependency.types.excluded	default: NONE Acceptable Values: NONE, BUILD Bitbake Excluded Dependency Types: The dependency types to exclude from the results.

blackduck-server

Property	Description
blackduck.api.token	Black Duck API Token: The access token used to authenticate with the Black Duck Server.
blackduck.offline.mode	default: false Offline Mode: This can disable any Black Duck communication - if true, Detect will not upload BDIO files, it will not check policies, and it will not download and install the signature scanner.
blackduck.url	Black Duck URL: URL of the Black Duck server.
detect.test.connection	default: false Test Connection to Black Duck: Test the connection to Black Duck with the current configuration.
blackduck.offline.mode.force.bdio	default: false Force Offline BDIO Generation: This property will force Detect in offline mode to generate a BDIO even if no code locations were identified.
blackduck.trust.cert (Advanced)	default: false Trust All SSL Certificates: If true, automatically trust the certificate for the current run of Detect only.
detect.timeout (Advanced)	default: 300 Detect Timeout: The amount of time in seconds Detect will wait for network connection, for scans to finish, and to generate reports (i.e. risk and policy check). When changing this value, keep in mind the checking of policies might have to wait for scans to process which can take some time.

Property	Description
	default: INTELLIGENT Acceptable Values: RAPID, EPHEMERAL, STATELESS, INTELLIGENT
detect.blackduck.scan.mode (Advanced)	Detect Scan Mode: Set the Black Duck scanning mode of Detect **DEPRECATED VALUES:** EPHEMERAL: Replace with STATELESS

cleanup

Property	Description
	default: true
detect.cleanup	Cleanup Output: If true, the files created by Detect will be cleaned up.

conan

Property	Description
detect.conan.path	Conan Executable: The path to the conan executable.
detect.conan.dependency.types.excluded	default: NONE Acceptable Values: NONE, BUILD Conan Dependency Types Excluded: Set this value to indicate which Conan dependency types Detect should exclude from the BOM.
detect.conan.arguments	Additional Conan Arguments: A space-separated list of additional arguments to add to the 'conan info' command line when running Detect against a Conan project. Detect will execute the command 'conan info {additional arguments} .'

Property	Description
detect.conan.lockfile.path	Conan Lockfile: The path to the conan lockfile to apply when running 'conan info' to get the dependency graph. If set, Detect will execute the command 'conan info --lockfile {lockfile} .'
detect.conan.attempt.package.revision.match	default: false Attempt Package Revision Match: If package revisions are available (a Conan lock file is found or provided, and Conan's revisions feature is enabled), require that each dependency's package revision match the package revision of the component in the KB.

conda

Property	Description
detect.conda.environment.name	Anaconda Environment Name: The name of the anaconda environment used by your project.
detect.conda.path	Conda Executable: The path to the conda executable.

cpan

Property	Description
detect.cpan.path	cpan Executable: The path to the cpan executable.
detect.cpanm.path	cpanm Executable: The path to the cpanm executable.

dart

Property	Description
detect.dart.path	dart Executable: The path to the dart executable.
detect.flutter.path	flutter Executable: The path to the flutter executable.
detect.pub.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Dart Pub Dependency Types Excluded: Set this value to indicate which Dart pub dependency types Detect should exclude from the BOM.

debug

Property	Description
detect.diagnostic	default: false Diagnostic Mode: When enabled, diagnostic mode collects files valuable for troubleshooting (logs, BDIO file, extraction files, reports, etc.), writes them to a zip file, and logs the path to the zip file.
detect.diagnostic.extended (Deprecated)	default: false Diagnostic Mode Extended: When enabled, Synopsys Detect performs the actions of --detect.diagnostic, but also includes relevant files such as lock files and build artifacts. **DEPRECATED: This property is being removed. Use property detect.diagnostic instead. There is no longer any distinction between extended and non-extended diagnostic zip files. This property will be removed in 9.0.0.**

default

Property	Description
detect.phone.home.passthrough (Advanced)	Phone Home Passthrough: Additional values may be sent home for usage information. The keys will be sent without the prefix.

detector

Property	Description
detect.required.detector.types	Acceptable Values: BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE Required Detect Types: The set of required detectors.
detect.accuracy.required (Advanced)	default: ALL Acceptable Values: ALL, NONE, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE Detector Accuracy Requirements: Detector types from which HIGH accuracy results are required when a detector of that type applies.

Property	Description
detect.excluded.detector.types (Advanced)	default: NONE Acceptable Values: NONE, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE Detector Types Excluded: By default, all detectors will be included. If you want to exclude specific detectors, specify the ones to exclude here. Exclusion rules always win.
detect.included.detector.types (Advanced)	default: ALL Acceptable Values: ALL, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE Detector Types Included: By default, all tools will be included. If you want to include only specific tools, specify the ones to include here. Exclusion rules always win.

docker

Property	Description
detect.docker.image	Docker Image Name: The Docker image name (repo:tag) to inspect.
detect.docker.image.id	Docker Image ID: The ID (shown in the 'IMAGE ID' column of 'docker images' output) of the target Docker image. The target image must already be local (must appear in the output of 'docker images').

Property	Description
detect.docker.path	Docker Executable: Path to the docker executable (used to load image inspector Docker images in order to run the Docker Inspector in air gap mode).
detect.docker.tar	Image Archive File: An image .tar file which is either a Docker image saved to a file using the 'docker save' command, or an Open Container Initiative (OCI) image .tar file. The file must be readable by all.
detect.docker.passthrough (Advanced)	Docker Passthrough: Additional properties may be passed to the docker inspector by adding the prefix detect.docker.passthrough to each Docker Inspector property name and assigning a value. The 'detect.docker.passthrough' prefix will be removed from the property name to generate the property name passed to Docker Inspector (with the given value).
detect.docker.inspector.path (Advanced)	Docker Inspector Path: Use this property to point Detect to a local Docker Inspector jar file, instead of the default Docker Inspector jar file that Detect downloads from the binary repository. You need to ensure the version is compatible (the same major version that Detect downloads by default).
detect.docker.inspector.version (Advanced)	Docker Inspector Version: Version of the Docker Inspector to use. By default Detect will attempt to automatically determine the version to use.
detect.docker.platform.top.layer.id (Advanced)	Platform Top Layer ID: To exclude components from platform layers from the results, assign to this property the ID of the top layer of the platform image. Get the platform top layer ID from the output of 'docker inspect platformimage:tag'. The platform top layer ID is the last item in RootFS.Layers. For more information, see 'Isolating application components' in the Docker Inspector documentation.

general

Property	Description
detect.target.type	default: SOURCE Acceptable Values: SOURCE, IMAGE Detect Target: Informs detect of what is being scanned which allows improved user experience when scanning different types of targets.
detect.wait.for.results	default: false Wait For Results: If set to true, Detect will wait for Synopsys products until results are available or the detect.timeout is exceeded.
detect.follow.symbolic.links	default: true Follow Symbolic Links: If set to true, Detect will follow symbolic links when searching for detectors, when searching for files that select detectors (such as Bitbake and Sbt) need, when searching for directories to exclude from signature scan, and when searching for binary scan targets. Symbolic links are not supported for Impact Analysis.
detect.parallel.processors (Advanced)	default: 1 Detect Parallel Processors: The number of threads to run processes in parallel, defaults to 1, but if you specify less than or equal to 0, the number of processors on the machine will be used.
detect.ignore.connection.failures (Advanced)	default: false Detect Ignore Connection Failures: If true, Detect will ignore any products (eg. Black Duck) that it cannot connect to.
detect.force.success (Advanced)	default: false Force Success: If true, Detect will always exit with code 0.

Property	Description
detect.force.success.on.skip (Advanced)	default: false Force Success On Skip: If true, Detect will always exit with code 0 when a scan of any type is skipped. Typically this happens when the Black Duck minimum scan interval timer has not been met.

go

Property	Description
detect.go.path	Go Executable: Path to the Go executable.
detect.go.mod.dependency.types.excluded	default: NONE Acceptable Values: NONE, UNUSED, VENDORED Go Mod Dependency Types Excluded: Set this value to indicate which Go Mod dependency types Detect should exclude from the BOM.

gradle

Property	Description
detect.gradle.build.command	Gradle Build Command: Gradle command line arguments to add to the gradle/gradlew command line.
detect.gradle.configuration.types.excluded	default: NONE Acceptable Values: NONE, UNRESOLVED Gradle Configuration Types Excluded: Set this value to indicate which Gradle configuration types Detect should exclude from the BOM.
detect.gradle.path	Gradle Executable: The path to the Gradle executable (gradle or gradlew).

Property	Description
detect.gradle.excluded.configurations (Advanced)	Gradle Exclude Configurations: A comma-separated list of Gradle configurations to exclude.
detect.gradle.excluded.projects (Advanced)	Gradle Exclude Projects: A comma-separated list of Gradle subprojects to exclude.
detect.gradle.excluded.project.paths (Advanced)	Gradle Exclude Subproject Paths: A comma-separated list of Gradle subproject paths to exclude.
detect.gradle.included.configurations (Advanced)	Gradle Include Configurations: A comma-separated list of Gradle configurations to include.
detect.gradle.included.projects (Advanced)	Gradle Include Projects: A comma-separated list of Gradle subprojects to include.
detect.gradle.included.project.paths (Advanced)	Gradle Include Project Paths: A comma-separated list of Gradle subproject paths to include.

hex

Property	Description
detect.hex.rebar3.path	Rebar3 Executable: The path to the rebar3 executable.

iac-scan

Property	Description
detect.iac.scan.paths	IaC Scan Target Paths: A comma-separated list of paths to perform IaC scans on.
detect.iac.scan.arguments	IaC Scan Arguments: A space-separated list of additional arguments to use when running the IaC Scanner.

Property	Description
detect.iac.scanner.local.path	IaC Scanner Local Path: Use this property to specify the path to a local IaC Scanner.

impact-analysis

Property	Description
detect.impact.analysis.enabled	default: false Vulnerability Impact Analysis Enabled: If set to true, Detect will attempt to look for *.class files and generate a Vulnerability Impact Analysis Report for upload to Black Duck.
detect.impact.analysis.output.path	Impact Analysis Output Directory: The path to the output directory for Impact Analysis reports.

lerna

Property	Description
detect.lerna.path	Lerna Executable: Path of the lerna executable.
detect.lerna.package.types.excluded	default: NONE Acceptable Values: NONE, PRIVATE Lerna Package Types Excluded: Set this value to indicate which Lerna package types Detect should exclude from the BOM.
detect.lerna.excluded.packages (Advanced)	Lerna Packages Excluded: A comma-separated list of Lerna packages to exclude.
detect.lerna.included.packages (Advanced)	Lerna Packages Included: A comma-separated list of Lerna packages to include.

logging

Property	Description
logging.level.com.synopsys.integration	default: INFO Acceptable Values: OFF, ERROR, WARN, INFO, DEBUG, TRACE Logging Level: The logging level of Detect.
logging.level.detect	default: INFO Acceptable Values: OFF, ERROR, WARN, INFO, DEBUG, TRACE Logging Level Shorthand: Shorthand for the logging level of detect. Equivalent to setting *logging.level.com.synopsys.integration*.

maven

Property	Description
detect.maven.build.command	Maven Build Command: Maven command line arguments to add to the mvn/mvnw command line.
detect.maven.path	Maven Executable: The path to the Maven executable (mvn or mvnw).
detect.maven.included.scopes	Dependency Scope Included: A comma separated list of Maven scopes. Output will be limited to dependencies within these scopes (overridden by exclude).
detect.maven.excluded.scopes	Dependency Scope Excluded: A comma separated list of Maven scopes. Output will be limited to dependencies outside these scopes (overrides include).
detect.maven.excluded.modules (Advanced)	Maven Modules Excluded: A comma-separated list of Maven modules (subprojects) to exclude.

Property	Description
detect.maven.included.modules (Advanced)	Maven Modules Included: A comma-separated list of Maven modules (subprojects) to include.

npm

Property	Description
detect.npm.arguments	Additional NPM Command Arguments: A space-separated list of additional arguments that Detect will add at then end of the npm ls command line when Detect executes the NPM CLI Detector on an NPM project.
detect.npm.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV, PEER Npm Dependency Types Excluded: Set this value to indicate which Npm dependency types Detect should exclude from the BOM.
detect.npm.path	NPM Executable: The path to the Npm executable.

nuget

Property	Description
detect.nuget.config.path	Nuget Config File: The path to the Nuget.Config file to supply to the nuget exe.
detect.nuget.packages.repo.url	default: https://api.nuget.org/v3/index.json Nuget Packages Repository URL: The source for nuget packages

Property	Description
detect.nuget.excluded.modules (Advanced)	Nuget Projects Excluded: The projects within the solution to exclude. Detect will exclude all projects with names that include any of the given regex patterns. To match a full project name (for example: 'BaGet.Core'), use a regular expression that matches only the full name ('^BaGet.Core$')
detect.nuget.ignore.failure (Advanced)	default: false Ignore Nuget Failures: If true errors will be logged and then ignored.
detect.nuget.included.modules (Advanced)	Nuget Modules Included: The names of the projects in a solution to include (overrides exclude). Detect will include all projects with names that include any of the given regex patterns. To match a full project name (for example: 'BaGet.Core'), use a regular expression that matches only the full name ('^BaGet.Core$')

packagist

Property	Description
detect.packagist.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Packagist Dependency Types Excluded: Set this value to indicate which Packagist dependency types Detect should exclude from the BOM.

paths

Property	Description
detect.bash.path	Bash Executable: Path to the Bash executable.
detect.bdio.output.path	BDIO Output Directory: The path to the output directory for the generated BDIO file.

Property	Description
detect.bdio.file.name	BDIO File Name: The desired file name of BDIO file Detect produces in the BDIO Output Directory.
detect.detector.search.depth	default: 0 Detector Search Depth: Depth of subdirectories within the source directory to which Detect will search for files that indicate whether a detector applies.
detect.git.path	Git Executable: Path of the git executable
detect.java.path	Java Executable: Path to the Java executable used by Docker Inspector.
detect.output.path	Detect Output Path: The path to the output directory.
detect.scan.output.path	Scan Output Path: The output directory for all signature scanner output files. If not set, the signature scanner output files will be in a 'scan' subdirectory of the output directory.
detect.source.path	Source Path: The source path is the path to the project directory to inspect. If no value is provided, the source path defaults to the current working directory.
detect.tools.excluded	Acceptable Values: NONE, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN Detect Tools Excluded: The tools Detect should not allow, in a comma-separated list. Excluded tools will not be run even if all criteria for the tool is met. Exclusion rules always win.

Property	Description
detect.tools	Acceptable Values: ALL, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN Detect Tools Included: The tools Detect should allow in a comma-separated list. Tools in this list (as long as they are not also in the excluded list) will be allowed to run if all criteria of the tool are met. Exclusion rules always win.
detect.status.json.output.path	Status JSON Output Path: The directory to place a copy of the status.json file.
detect.scaaas.scan.path	SCAAAS Scan Target: This file will be uploaded to the BDBA worker for scan analysis in an SCA as a service environment.
detect.detector.search.continue (Advanced)	default: false Detector Search Continue: By default, nesting rules limit which detectors can run on a subdirectory based on which detectors applied on any parent directory. Setting this property to true disables nesting rules.
detect.excluded.directories (Advanced)	Detect Excluded Directories: A comma-separated list of names, name patterns, relative paths, or path patterns of directories that Detect should exclude.
detect.excluded.directories.defaults.disabled (Advanced)	default: false Detect Excluded Directories Defaults Disabled: If false, Detect will exclude the default directory names. See the detailed help for more information.
detect.tools.output.path (Advanced)	Detect Tools Output Path: The path to the tools directory where detect should download and/or access things like the Signature Scanner that it shares over multiple runs.

Property	Description
detect.project.detector (Advanced)	Project Name and Version Detector: The detector that will be used to determine the project name and version when multiple detector types apply. This property should be used with detect.project.tool.
detect.project.tool (Advanced)	default: DOCKER,DETECTOR,BAZEL Acceptable Values: DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN Detector Tool Priority: The tool priority for project name and version. The project name and version will be determined by the first tool in this list that provides them.

pear

Property	Description
detect.pear.dependency.types.excluded	default: NONE Acceptable Values: NONE, OPTIONAL Pear Dependency Types Excluded: Set this value to indicate which Pear dependency types Detect should exclude from the BOM.
detect.pear.path	Pear Executable: The path to the pear executable.

pip

Property	Description
detect.pipfile.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV Pipfile Dependency Types Excluded: A comma-separated list of dependency types that will be excluded.
detect.pip.project.name	PIP Project Name: The name of your PIP project, to be used if your project's name cannot be correctly inferred from its setup.py file.
detect.pip.project.version.name	PIP Project Version Name: The version of your PIP project, to be used if your project's version name cannot be correctly inferred from its setup.py file.
detect.pip.requirements.path	PIP Requirements Path: A comma-separated list of paths to requirements files, to be used to analyze requirements files with a filename other than requirements.txt or to specify which requirements files should be analyzed.
detect.pip.only.project.tree	default: false PIP Include Only Project Tree: By default, pipenv includes all dependencies found in the graph. Set to true to only include dependencies found underneath the dependency that matches the provided pip project and version name.
detect.pip.path	Pip Executable: The path to the Pip executable.
detect.pipenv.path	Pipenv Executable: The path to the Pipenv executable.

pnpm

Property	Description
detect.pnpm.dependency.types.excluded	default: NONE Acceptable Values: NONE, DEV, OPTIONAL pnpm Dependency Types: Set this value to indicate which pnpm dependency types Detect should exclude from the BOM.

project

Property	Description
detect.policy.check.fail.on.severities	default: NONE Acceptable Values: ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED Fail on Policy Violation Severities: A comma-separated list of policy violation severities that will fail Detect. If this is set to NONE, Detect will not fail due to policy violations. A value of ALL is equivalent to all of the other possible values except NONE.
detect.policy.check.fail.on.names	Fail on Policy Names with Violations: A comma-separated list of policy names with a non-zero number of violations that will fail Detect.
detect.project.description	Project Description: If project description is specified, your project will be created with this description. For updates, see detect.project.version.update.
detect.project.name	Project Name: An override for the name to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable project name. If that fails, the final part of the directory path where the inspection is taking place will be used.

Property	Description
detect.project.tier	Project Tier: If a Black Duck project tier is specified, your project will be created with this tier. For updates, see detect.project.version.update.
detect.project.version.name	Version Name: An override for the version to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable version name. If that fails, the current date will be used.
detect.project.version.nickname	Version Nickname: If a project version nickname is specified, your project version will be created with this nickname. For updates, see detect.project.version.update.
detect.project.version.notes	Version Notes: If project version notes are specified, your project version will be created with these notes. For updates, see detect.project.version.update.
detect.project.version.phase	default: DEVELOPMENT Acceptable Values: ARCHIVED, DEPRECATED, DEVELOPMENT, PLANNING, PRERELEASE, RELEASED Version Phase: If project version phase is specified, your project version will be created with this phase. For updates, see detect.project.version.update.
detect.project.version.update	default: false Update Project Version: If set to true, Detect will update the Black Duck project and project version according to configured project and project version properties. (By default, these properties are only set on created projects / project versions.)
detect.project.version.license	Project Version License: If project version license is specified, your project version will be created with this license. For updates, see detect.project.version.update.

Property	Description
detect.clone.project.version.name (Advanced)	Clone Project Version Name: The name of the project version to clone this project version from. Respects the given Clone Categories in detect.project.clone.categories or as set on the Black Duck server.
detect.clone.project.version.latest (Advanced)	default: false Clone Latest Project Version: If set to true, detect will attempt to use the latest project version as the clone for this project. The project must exist and have at least one version.
detect.code.location.name (Advanced)	Scan Name: An override for the base name Detect will use for the scan (codelocation) it creates. Detect appends a suffix to the base name that indicates the source ("scan" for the signature scanner, "gradle/bom" for the Gradle detector, etc.). If this property is set and multiple code locations are generated from the same source, Detect will also append an index to avoid name collisions. When this property is set, detect.project.codelocation.prefix and detect.project.codelocation.suffix are ignored.
detect.project.application.id (Advanced)	Application ID: Sets the 'Application ID' project setting.
detect.project.group.name (Advanced)	Project Group Name: Sets the 'Project Group' to assign the project to. Must match exactly to an existing project group on Black Duck.
detect.custom.fields.project (Advanced)	Custom Fields: A list of custom fields with a label and comma-separated value starting from index 0. For each index, provide one label and one value. For example, to set a custom field with label 'example' to 'one,two': `detect.custom.fields.project[0].label='example'` and `detect.custom.fields.project[0].value='one,two'`. To set another field, use index 1. Note that these will not show up in the detect configuration log.

Property	Description
detect.custom.fields.version (Advanced)	Custom Fields: A list of custom fields with a label and comma-separated value starting from index 0. For each index, provide one label and one value. For example , to set a custom field with label 'example' to 'one,two': `detect.custom.fields.version[0].label='example'` and `detect.custom.fields.version[0].value='one,two'`. To set another field, use index 1. Note that these will not show up in the detect configuration log.
detect.project.clone.categories (Advanced)	default: ALL Acceptable Values: ALL, NONE, COMPONENT_DATA, CUSTOM_FIELD_DATA, DEEP_LICENSE, LICENSE_TERM_FULFILLMENT, VERSION_SETTINGS, VULN_DATA Clone Project Categories: The value of this property is used to set the 'Cloning' settings on created Black Duck projects. If property detect.project.version.update is set to true, the value of this property is used to set the 'Cloning' settings on updated Black Duck projects.
detect.project.codelocation.prefix (Advanced)	Scan Name Prefix: A prefix to the name of the scans created by Detect. Useful for running against the same projects on multiple machines.
detect.project.codelocation.suffix (Advanced)	Scan Name Suffix: A suffix to the name of the scans created by Detect.
detect.project.codelocation.unmap (Advanced)	default: false Unmap All Other Scans for Project: If set to true, unmaps all other scans mapped to the project version produced by the current run of Detect.
detect.project.user.groups (Advanced)	Project User Groups: A comma-separated list of names of user groups to add to the project.
detect.project.tags (Advanced)	Project Tags: A comma-separated list of tags to add to the project. This property is not supported when using Synopsys Detect in offline mode.

Property	Description
detect.project.level.adjustments (Advanced)	default: true Allow Project Level Adjustments: If set, created projects will be created with the value of this property. For updates, see detect.project.version.update.
detect.parent.project.name (Advanced)	Parent Project Name: When a parent project and version name are specified, the created detect project will be added as a component to the specified parent project version. The specified parent project and parent project version must exist on Black Duck.
detect.parent.project.version.name (Advanced)	Parent Project Version Name: When a parent project and version name are specified, the created detect project will be added as a component to the specified parent project version. The specified parent project and parent project version must exist on Black Duck.
detect.project.version.distribution (Advanced)	default: EXTERNAL Acceptable Values: EXTERNAL, INTERNAL, OPENSOURCE, SAAS Version Distribution: If project version distribution is specified, your project version will be created with this distribution. For updates, see detect.project.version.update.

project-inspector

Property	Description
detect.project.inspector.path (Advanced)	Project Inspector Path: Use this property to point Detect to a local Project Inspector zip file, instead of the default Project Inspector zip file that Detect downloads from the binary repository. You need to ensure the version is compatible (the same major version that Detect downloads by default).

Property	Description
detect.project.inspector.global.arguments (Advanced)	Project Inspector Global Arguments: A space-separated list of global options to pass to all invocations of the project inspector.
detect.project.inspector.arguments (Advanced)	Project Inspector Additional Arguments: A space-separated list of additional options to pass to all invocations of the project inspector.

proxy

Property	Description
blackduck.proxy.host (Advanced)	Proxy Host: Hostname of the proxy server.
blackduck.proxy.ignored.hosts (Advanced)	Bypass Proxy Hosts: A comma separated list of regular expression host patterns that should not use the proxy.
blackduck.proxy.ntlm.domain (Advanced)	NTLM Proxy Domain: NTLM Proxy domain.
blackduck.proxy.ntlm.workstation (Advanced)	NTLM Proxy Workstation: NTLM Proxy workstation.
blackduck.proxy.password (Advanced)	Proxy Password: Proxy password.
blackduck.proxy.port (Advanced)	Proxy Port: Proxy port number.
blackduck.proxy.username (Advanced)	Proxy Username: Proxy username.

python

Property	Description
detect.python.path	Python Executable: The path to the Python executable.

rapid-scan

Property	Description
detect.blackduck.rapid.compare.mode (Advanced)	default: ALL Acceptable Values: ALL, BOM_COMPARE, BOM_COMPARE_STRICT Rapid Compare Mode: Controls how rapid scan evaluates policy rules

report

Property	Description
detect.notices.report	default: false Generate Notices Report: When set to true, a Black Duck notices report in text form will be created in your source directory.
detect.notices.report.path	Notices Report Path: The output directory for notices report. Default is the source directory.
detect.risk.report.pdf	default: false Generate Risk Report (PDF): When set to true, a Black Duck risk report in PDF form will be created.
detect.risk.report.pdf.path	Risk Report Output Path: The output directory for risk report in PDF. Default is the source directory.

ruby

Property	Description
detect.ruby.dependency.types.excluded	default: NONE Acceptable Values: NONE, RUNTIME, DEV Ruby Dependency Types Excluded: Set this value to indicate which Ruby(Gempsec) dependency types Detect should exclude from the BOM.

sbt

Property	Description
detect.sbt.path	Sbt Executable: Path to the Sbt executable.
detect.sbt.arguments	Additional sbt command Arguments: A space-separated list of additional arguments to add to sbt command line when running Detect against an SBT project. Detect will execute the command 'sbt {additional arguments} {Detect-added arguments}'.

signature-scanner

Property	Description
detect.blackduck.signature.scanner.arguments	Signature Scanner Arguments: A space-separated list of additional arguments to use when running the Black Duck signature scanner.
detect.blackduck.signature.scanner.copyright.search	default: false Signature Scanner Copyright Search: When set to true, user will be able to scan and discover copyright names in Black Duck. Corresponding Signature Scanner CLI Argument: --copyright-search.

Property	Description
detect.blackduck.signature.scanner.dry.run	default: false Signature Scanner Dry Run: If set to true, the signature scanner results are not uploaded to Black Duck, and the scanner results are written to disk via the Signature Scanner CLI argument: --dryRunWriteDir.
detect.blackduck.signature.scanner.individual.file.matching	default: NONE Acceptable Values: NONE, SOURCE, BINARY, ALL Individual File Matching: Users may set this property to indicate what types of files they want to match. Corresponding Signature Scanner CLI Argument: --individualFileMatching.
detect.blackduck.signature.scanner.license.search	default: false Signature Scanner License Search: When set to true, user will be able to scan and discover license names in Black Duck. Corresponding Signature Scanner CLI Argument: --license-search.
detect.blackduck.signature.scanner.local.path	Signature Scanner Local Path: To use a local signature scanner, specify the path where the signature scanner was unzipped. This will likely look similar to 'scan.cli-x.y.z' and includes the 'bin, icon, jre, and lib' directories of the expanded scan.cli.
detect.blackduck.signature.scanner.paths	Signature Scanner Target Paths: If this property is not set, the signature scanner target path is the source path (see property detect.source.path). If this property is set, the paths provided in this property's value will be signature scanned instead (the signature scanner will be executed once for each provided path).

Property	Description
detect.blackduck.signature.scanner.snippet.matching	default: NONE Acceptable Values: NONE, SNIPPET_MATCHING, SNIPPET_MATCHING_ONLY, FULL_SNIPPET_MATCHING, FULL_SNIPPET_MATCHING_ONLY Snippet Matching: Use this value to enable the various snippet scanning modes. For a full explanation, please refer to the 'Running a component scan using the Signature Scanner command line' section in your Black Duck server's online help. Corresponding Signature Scanner CLI Arguments: --snippet-matching, --snippet-matching-only, --full-snippet-scan.
detect.blackduck.signature.scanner.reduced.persistence	default: DEFAULT Acceptable Values: DEFAULT, RETAIN_UNMATCHED, DISCARD_UNMATCHED Reduced Persistence: Use this value to control how unmatched files from signature scans are stored. For a full explanation, please refer to *about reduced persistence signature scanning.*
detect.blackduck.signature.scanner.upload.source.mode	default: false Upload source mode: If set to true, the signature scanner will, if supported by your Black Duck version, upload source code to Black Duck. Corresponding Signature Scanner CLI Argument: --upload-source.
detect.excluded.directories.search.depth	default: 4 Detect Excluded Directories Search Depth: Enables you to adjust the depth to which Detect will search when creating signature scanner exclusion patterns.
detect.blackduck.signature.scanner.memory (Advanced)	default: 4096 Signature Scanner Memory: The memory for the scanner to use.

swift

Property	Description
detect.swift.path	Swift Executable: Path of the swift executable.

yarn

Property	Description
detect.yarn.dependency.types.excluded	default: NONE Acceptable Values: NONE, NON_PRODUCTION Yarn Dependency Types Excluded: Set this value to indicate which Yarn dependency types Detect should exclude from the BOM.
detect.yarn.excluded.workspaces (Advanced)	Yarn Exclude Workspaces: A comma-separated list of Yarn workspaces (specified by the workspace directory's relative path) to exclude.
detect.yarn.included.workspaces (Advanced)	Yarn Include Workspaces: A comma-separated list of Yarn workspaces (specified by the workspace directory's relative path) to include.

Documentation version: 8.10.0

Configuration Property Details

This section contains additional details about properties that configure Synopsys Detect execution.

binary-scanner

Binary Scan Filename Patterns

```
--detect.binary.scan.file.name.patterns
```

If specified, files in the source directory whose names match these file name patterns will be zipped and uploaded for binary scan analysis. This property will not be used if detect.binary.scan.file.path is specified. Search depth is controlled by property detect.binary.scan.search.depth. Directories

specified via property detect.excluded.directories are excluded from the file search. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Details	
Added	6.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`*.jar`

Binary Scan Search Depth

```
--detect.binary.scan.search.depth=0
```

When binary scan filename patterns are being used to search for binary files to scan, this property sets the depth at which Detect will search for files (that match those patterns) to upload for binary scan analysis.

Details	
Added	6.9.0
Type	Integer
Default Value	0
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Binary Scan Target

```
--detect.binary.scan.file.path
```

If specified, this file and this file only will be uploaded for binary scan analysis. This property takes precedence over detect.binary.scan.file.name.patterns. The BINARY_SCAN tool does not provide project and version name defaults to Detect, so you need to set project and version names via properties when only the BINARY_SCAN tool is invoked.

Details	
Added	4.2.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

blackduck-server

Black Duck API Token

```
--blackduck.api.token
```

The access token used to authenticate with the Black Duck Server.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

Black Duck URL

```
--blackduck.url
```

URL of the Black Duck server.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`https://blackduck.mydomain.com`

Force Offline BDIO Generation

```
--blackduck.offline.mode.force.bdio=false
```

This property will force Detect in offline mode to generate a BDIO even if no code locations were identified.

Details	
Added	8.5.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No

Details	
Acceptable Values	Any
Strict	No

Offline Mode

```
--blackduck.offline.mode=false
```

This can disable any Black Duck communication - if true, Detect will not upload BDIO files, it will not check policies, and it will not download and install the signature scanner.

Details	
Added	4.2.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Test Connection to Black Duck

```
--detect.test.connection=false
```

Test the connection to Black Duck with the current configuration.

Details	
Added	3.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No

Details	
Acceptable Values	Any
Strict	No

Detect Scan Mode (Advanced)

```
--detect.blackduck.scan.mode=RAPID,EPHEMERAL,STATELESS,INTELLIGENT
```

Set the Black Duck scanning mode of Detect

Set the scanning mode of Detect to control how Detect will send data to Black Duck. RAPID will not persist the results and disables select Detect functionality for faster results. INTELLIGENT persists the results and permits all features of Detect

Details	
Added	6.9.0
Type	BlackduckScanMode
Default Value	INTELLIGENT
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	RAPID, EPHEMERAL, STATELESS, INTELLIGENT
Strict	Yes
Deprecated Values	EPHEMERAL: Replace with STATELESS

Detect Timeout (Advanced)

```
--detect.timeout=300
```

The amount of time in seconds Detect will wait for network connection, for scans to finish, and to generate reports (i.e. risk and policy check). When changing this value, keep in mind the checking of policies might have to wait for scans to process which can take some time.

Details	
Added	6.8.0

Details	
Type	Long
Default Value	300
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`600`

Trust All SSL Certificates (Advanced)

```
--blackduck.trust.cert=false
```

If true, automatically trust the certificate for the current run of Detect only.

Details	
Added	4.2.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

cleanup

Cleanup Output

```
--detect.cleanup=true
```

If true, the files created by Detect will be cleaned up.

Details	
Added	3.2.0
Type	Boolean
Default Value	true
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

debug

Diagnostic Mode

```
--detect.diagnostic=false
```

When enabled, diagnostic mode collects files valuable for troubleshooting (logs, BDIO file, extraction files, reports, etc.), writes them to a zip file, and logs the path to the zip file.

Details	
Added	6.5.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Diagnostic Mode Extended (Deprecated)

```
--detect.diagnostic.extended=false
```

When enabled, Synopsys Detect performs the actions of --detect.diagnostic, but also includes relevant files such as lock files and build artifacts.

DEPRECATED: This property is being removed. Use property detect.diagnostic instead. There is no longer any distinction between extended and non-extended diagnostic zip files. This property will be removed in 9.0.0.

Details	
Added	6.5.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

default

Phone Home Passthrough (Advanced)

```
--detect.phone.home.passthrough
```

Additional values may be sent home for usage information. The keys will be sent without the prefix.

Details	
Added	6.0.0
Type	None
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

detector

Required Detect Types

```
--detect.required.detector.types=BITBAKE,CARGO,CARTHAGE,COCOAPODS,CONAN
,CONDA,CPAN,CRAN,DART,GIT,GO_MOD,GO_DEP,GO_VNDR,GO_VENDOR,GO_GRADLE,GRA
DLE,HEX,IVY,LERNA,MAVEN,NPM,NUGET,PACKAGIST,PEAR,PIP,PNPM,POETRY,RUBYGE
MS,SBT,SWIFT,YARN,CLANG,XCODE
```

The set of required detectors.

If you want one or more detectors to be required (must be found to apply), use this property to specify the set of required detectors. If this property is set, and one (or more) of the given detectors is not found to apply, Detect will fail.

Details	
Added	4.3.0
Type	DetectorType List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE
Strict	Yes
Example	`NPM`

Detector Accuracy Requirements (Advanced)

```
--detect.accuracy.required=ALL,NONE,BITBAKE,CARGO,CARTHAGE,COCOAPODS,CO
NAN,CONDA,CPAN,CRAN,DART,GIT,GO_MOD,GO_DEP,GO_VNDR,GO_VENDOR,GO_GRADLE,
```

```
GRADLE,HEX,IVY,LERNA,MAVEN,NPM,NUGET,PACKAGIST,PEAR,PIP,PNPM,POETRY,RUB
YGEMS,SBT,SWIFT,YARN,CLANG,XCODE
```

Detector types from which HIGH accuracy results are required when a detector of that type applies.

The value of this property only affects detector types which apply to the source project. If a detector type applies, and is one of the accuracy-required detector types indicated by the value of this property, low accuracy results for that detector type are treated as a failure. Refer to the *Downloading and Running Synopsys Detect > Detector search and accuracy* page for more details.

Details	
Added	7.13.0
Type	DetectorType List
Default Value	ALL
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	ALL, NONE, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE
Strict	Yes
Example	`ALL,NONE`

Detector Types Excluded (Advanced)

```
--detect.excluded.detector.types=NONE,BITBAKE,CARGO,CARTHAGE,COCOAPODS,
CONAN,CONDA,CPAN,CRAN,DART,GIT,GO_MOD,GO_DEP,GO_VNDR,GO_VENDOR,GO_GRADL
E,GRADLE,HEX,IVY,LERNA,MAVEN,NPM,NUGET,PACKAGIST,PEAR,PIP,PNPM,POETRY,R
UBYGEMS,SBT,SWIFT,YARN,CLANG,XCODE
```

By default, all detectors will be included. If you want to exclude specific detectors, specify the ones to exclude here. Exclusion rules always win.

If Detect runs one or more detector on your project that you would like to exclude, you can use this property to prevent Detect from running them.

Details	
Added	3.0.0
Type	DetectorType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE
Strict	Yes
Example	`NPM,LERNA`

Detector Types Included (Advanced)

```
--detect.included.detector.types=ALL,BITBAKE,CARGO,CARTHAGE,COCOAPODS,CONAN,CONDA,CPAN,CRAN,DART,GIT,GO_MOD,GO_DEP,GO_VNDR,GO_VENDOR,GO_GRADLE,GRADLE,HEX,IVY,LERNA,MAVEN,NPM,NUGET,PACKAGIST,PEAR,PIP,PNPM,POETRY,RUBYGEMS,SBT,SWIFT,YARN,CLANG,XCODE
```

By default, all tools will be included. If you want to include only specific tools, specify the ones to include here. Exclusion rules always win.

If you want to limit Detect to a subset of its detectors, use this property to specify that subset.

Details	
Added	3.0.0
Type	DetectorType List
Default Value	ALL
Comma Separated	Yes
Case Sensitive	No

Details	
Acceptable Values	ALL, BITBAKE, CARGO, CARTHAGE, COCOAPODS, CONAN, CONDA, CPAN, CRAN, DART, GIT, GO_MOD, GO_DEP, GO_VNDR, GO_VENDOR, GO_GRADLE, GRADLE, HEX, IVY, LERNA, MAVEN, NPM, NUGET, PACKAGIST, PEAR, PIP, PNPM, POETRY, RUBYGEMS, SBT, SWIFT, YARN, CLANG, XCODE
Strict	Yes
Example	`NPM`

general

Detect Target

```
--detect.target.type=SOURCE,IMAGE
```

Informs detect of what is being scanned which allows improved user experience when scanning different types of targets.

Changes the behaviour of detect to better suite what is being scanned. For example, when IMAGE is selected and the DOCKER tool applies and has not been excluded, detect will not pick a source directory, will automatically disable the DETECTOR tool and run BINARY/SIGNATURE SCAN on the provided image.

Details	
Added	7.0.0
Type	DetectTargetType
Default Value	SOURCE
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	SOURCE, IMAGE
Strict	Yes

Follow Symbolic Links

```
--detect.follow.symbolic.links=true
```

If set to true, Detect will follow symbolic links when searching for detectors, when searching for files that select detectors (such as Bitbake and Sbt) need, when searching for directories to exclude from signature scan, and when searching for binary scan targets. Symbolic links are not supported for Impact Analysis.

Details	
Added	7.0.0
Type	Boolean
Default Value	true
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Wait For Results

```
--detect.wait.for.results=false
```

If set to true, Detect will wait for Synopsys products until results are available or the detect.timeout is exceeded.

Details	
Added	5.5.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detect Ignore Connection Failures (Advanced)

```
--detect.ignore.connection.failures=false
```

If true, Detect will ignore any products (eg. Black Duck) that it cannot connect to.

If true, when Detect attempts to boot a product (eg. Black Duck) it will also check if it can communicate with it - if it cannot, it will not run the product.

Details	
Added	5.3.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detect Parallel Processors (Advanced)

```
--detect.parallel.processors=1
```

The number of threads to run processes in parallel, defaults to 1, but if you specify less than or equal to 0, the number of processors on the machine will be used.

Details	
Added	6.0.0
Type	Integer
Default Value	1
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Force Success (Advanced)

```
--detect.force.success=false
```

If true, Detect will always exit with code 0.

Details	
Added	3.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Force Success On Skip (Advanced)

```
--detect.force.success.on.skip=false
```

If true, Detect will always exit with code 0 when a scan of any type is skipped. Typically this happens when the Black Duck minimum scan interval timer has not been met.

Details	
Added	7.12.1
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

impact-analysis

Impact Analysis Output Directory

```
--detect.impact.analysis.output.path
```

The path to the output directory for Impact Analysis reports.

If not set, the Impact Analysis reports are placed in a 'impact-analysis' subdirectory of the output directory.

Details	
Added	6.5.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Vulnerability Impact Analysis Enabled

```
--detect.impact.analysis.enabled=false
```

If set to true, Detect will attempt to look for *.class files and generate a Vulnerability Impact Analysis Report for upload to Black Duck.

Details	
Added	6.5.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

logging

Logging Level

```
--logging.level.com.synopsys.integration=OFF,ERROR,WARN,INFO,DEBUG,TRAC
E
```

The logging level of Detect.

To keep the log file size manageable, use INFO level logging for normal use. Use DEBUG or TRACE for troubleshooting.
Detect logging uses Spring Boot logging, which uses Logback (https://logback.qos.ch). The format of this property name is *logging.level.{package}[.{class}]*. The property name shown above specifies package *com.synopsys.integration* because that is the name of Detect's top-level package. Changing the logging level for that package changes the logging level for all Detect code, as well as Synopsys integration libraries that Detect uses. Non-Synopsys libraries that Detect uses are not affected. However, you can use this property to set the logging level for some of the non-Synopsys libraries that Detect uses by using the appropriate package name. For example, *logging.level.org.apache.http=TRACE* sets the logging level to TRACE for the Apache HTTP client library.
For log message format, Detect uses a default value of *%d{yyyy-MM-dd HH:mm:ss z} ${LOG_LEVEL_PATTERN:%-6p}[%thread] %clr(---){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:%wEx}*. You can change your log message format by setting the Spring Boot *logging.pattern.console* property to a different pattern.
Refer to the Spring Boot logging and Logback Project documentation for more details.

Details	
Added	5.3.0
Type	LogLevel
Default Value	INFO
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	OFF, ERROR, WARN, INFO, DEBUG, TRACE
Strict	Yes

Logging Level Shorthand

```
--logging.level.detect=OFF,ERROR,WARN,INFO,DEBUG,TRACE
```

Shorthand for the logging level of detect. Equivalent to setting *logging.level.com.synopsys.integration*.

Refer to the description of property *logging.level.com.synopsys.integration* for additional details.

Details	
Added	5.5.0
Type	LogLevel
Default Value	INFO
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	OFF, ERROR, WARN, INFO, DEBUG, TRACE
Strict	Yes

paths

Bash Executable

```
--detect.bash.path
```

Path to the Bash executable.

If set, Detect will use the given Bash executable instead of searching for one.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No
Example	`/usr/bin/bash`

BDIO File Name

```
--detect.bdio.file.name
```

The desired file name of BDIO file Detect produces in the BDIO Output Directory.

If not set, the file name is generated from your project, version and code location names.

Details	
Added	7.9.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

BDIO Output Directory

```
--detect.bdio.output.path
```

The path to the output directory for the generated BDIO file.

If not set, the BDIO file will be placed in a 'BDIO' subdirectory of the output directory.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No

Details	
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detector Search Depth

```
--detect.detector.search.depth=0
```

Depth of subdirectories within the source directory to which Detect will search for files that indicate whether a detector applies.

A value of 0 (the default) tells Detect not to search any subdirectories, a value of 1 tells Detect to search first-level subdirectories, etc.

Details	
Added	3.2.0
Type	Integer
Default Value	0
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detect Output Path

```
--detect.output.path
```

The path to the output directory.

If set, Detect will use the given directory to store files that it downloads and creates, instead of using the default location (~/blackduck).

Details	
Added	3.0.0

Details	
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detect Tools Excluded

```
--detect.tools.excluded=NONE,DETECTOR,SIGNATURE_SCAN,BINARY_SCAN,IMPACT
_ANALYSIS,DOCKER,BAZEL,IAC_SCAN,CONTAINER_SCAN
```

The tools Detect should not allow, in a comma-separated list. Excluded tools will not be run even if all criteria for the tool is met. Exclusion rules always win.

This property and detect.tools provide control over which tools Detect runs. If neither detect.tools nor detect.tools.excluded are set, Detect will allow (run if applicable, based on the values of other properties) all Detect tools. If detect.tools is set, and detect.tools.excluded is not set, Detect will only allow to run those tools that are specified in the detect.tools list. If detect.tools.excluded is set, Detect will only allow those tools that are not specified in the detect.tools.excluded list.

Details	
Added	5.0.0
Type	DetectTool List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN
Strict	Yes

Detect Tools Included

```
--detect.tools=ALL,DETECTOR,SIGNATURE_SCAN,BINARY_SCAN,IMPACT_ANALYSIS,
DOCKER,BAZEL,IAC_SCAN,CONTAINER_SCAN
```

The tools Detect should allow in a comma-separated list. Tools in this list (as long as they are not also in the excluded list) will be allowed to run if all criteria of the tool are met. Exclusion rules always win.

This property and detect.tools.excluded provide control over which tools Detect runs. If neither detect.tools nor detect.tools.excluded are set, Detect will allow (run if applicable, based on the values of other properties) all Detect tools. If detect.tools is set, and detect.tools.excluded is not set, Detect will only allow to run those tools that are specified in the detect.tools list. If detect.tools.excluded is set, Detect will only allow those tools that are not specified in the detect.tools.excluded list.

Details	
Added	5.0.0
Type	DetectTool List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	ALL, DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN
Strict	Yes

Git Executable

```
--detect.git.path
```

Path of the git executable

Details	
Added	5.5.0

Details	
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Java Executable

```
--detect.java.path
```

Path to the Java executable used by Docker Inspector.

If set, Detect will use the given Java executable instead of searching for one.

Details	
Added	5.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

SCAAAS Scan Target

```
--detect.scaaas.scan.path
```

This file will be uploaded to the BDBA worker for scan analysis in an SCA as a service environment.

Details	
Added	8.8.0

Details	
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Scan Output Path

```
--detect.scan.output.path
```

The output directory for all signature scanner output files. If not set, the signature scanner output files will be in a 'scan' subdirectory of the output directory.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Source Path

```
--detect.source.path
```

The source path is the path to the project directory to inspect. If no value is provided, the source path defaults to the current working directory.

Detect will search the source directory for hints that indicate which package manager(s) the project uses, and will attempt to run the corresponding detector(s).The source path is also the default target

for signature scanning. (This can be overridden with the detect.blackduck.signature.scanner.paths property.)

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Status JSON Output Path

```
--detect.status.json.output.path
```

The directory to place a copy of the status.json file.

If set, Detect will use the given directory to store a copy of the status.json file.

Details	
Added	8.1.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detect Excluded Directories (Advanced)

```
--detect.excluded.directories
```

A comma-separated list of names, name patterns, relative paths, or path patterns of directories that Detect should exclude.

Subdirectories whose name or path is resolved from the patterns in this list will not be searched when determining which detectors to run, will not be searched to find files for binary scanning when property detect.binary.scan.file.name.patterns is set, and will be excluded from signature scan using the Scan CLI '--exclude' flag. Refer to the *Downloading and Running Synopsys Detect > Including and Excluding Tools, Detectors, Directories, etc.* page for more details.

Details	
Added	7.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`**/*-test`

Detect Excluded Directories Defaults Disabled (Advanced)

```
--detect.excluded.directories.defaults.disabled=false
```

If false, Detect will exclude the default directory names. See the detailed help for more information.

If false, the following directories will be excluded by Detect when searching for detectors: __MACOX, bin, build, .git, .gradle, .yarn, node_modules, out, packages, target, .synopsys, and the following directories will be excluded from signature scan using the Scan CLI '--exclude' flag: .git, .gradle, node_modules, .synopsys.

Details	
Added	7.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

Detector Search Continue (Advanced)

```
--detect.detector.search.continue=false
```

By default, nesting rules limit which detectors can run on a subdirectory based on which detectors applied on any parent directory. Setting this property to true disables nesting rules.

Refer to the 'Downloading and Running Synopsys Detect' > 'Detector search and accuracy' page for more information on nesting rules.

Details	
Added	3.2.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detector Tool Priority (Advanced)

```
--detect.project.tool=DETECTOR,SIGNATURE_SCAN,BINARY_SCAN,IMPACT_ANALYS
IS,DOCKER,BAZEL,IAC_SCAN,CONTAINER_SCAN
```

The tool priority for project name and version. The project name and version will be determined by the first tool in this list that provides them.

This allows you to control which tool provides the project name and version when more than one tool are capable of providing it.

Details	
Added	5.0.0
Type	DetectTool List

Details	
Default Value	DOCKER,DETECTOR,BAZEL
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	DETECTOR, SIGNATURE_SCAN, BINARY_SCAN, IMPACT_ANALYSIS, DOCKER, BAZEL, IAC_SCAN, CONTAINER_SCAN
Strict	Yes

Detect Tools Output Path (Advanced)

```
--detect.tools.output.path
```

The path to the tools directory where detect should download and/or access things like the Signature Scanner that it shares over multiple runs.

If set, Detect will use the given directory instead of using the default location of output path plus tools.

Details	
Added	5.6.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Project Name and Version Detector (Advanced)

```
--detect.project.detector
```

The detector that will be used to determine the project name and version when multiple detector types apply. This property should be used with detect.project.tool.

If Detect finds that multiple detectors apply, this property can be used to select the detector that will provide the project name and version. When using this property, you should also set detect.project.tool=DETECTOR

Details	
Added	4.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

project

Fail on Policy Names with Violations

```
--detect.policy.check.fail.on.names
```

A comma-separated list of policy names with a non-zero number of violations that will fail Detect.

If left unset, Detect will not fail due to violated policies of a certain name. This property does not change the behavior of detect.policy.check.fail.on.severities.

Details	
Added	7.12.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No

Fail on Policy Violation Severities

```
--detect.policy.check.fail.on.severities=ALL,NONE,BLOCKER,CRITICAL,MAJO
R,MINOR,OK,TRIVIAL,UNSPECIFIED
```

A comma-separated list of policy violation severities that will fail Detect. If this is set to NONE, Detect will not fail due to policy violations. A value of ALL is equivalent to all of the other possible values except NONE.

Details	
Added	3.0.0
Type	PolicyRuleSeverityType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED
Strict	Yes

Project Description

```
--detect.project.description
```

If project description is specified, your project will be created with this description. For updates, see detect.project.version.update.

Details	
Added	4.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

Project Name

```
--detect.project.name
```

An override for the name to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable project name. If that fails, the final part of the directory path where the inspection is taking place will be used.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Project Tier

```
--detect.project.tier
```

If a Black Duck project tier is specified, your project will be created with this tier. For updates, see detect.project.version.update.

Details	
Added	3.1.0
Type	Optional Integer
Default Value	
Comma Separated	No
Case Sensitive	No

Details	
Acceptable Values	Any
Strict	No

Project Version License

```
--detect.project.version.license
```

If project version license is specified, your project version will be created with this license. For updates, see detect.project.version.update.

Details	
Added	7.11.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`Apache License 2.0`

Update Project Version

```
--detect.project.version.update=false
```

If set to true, Detect will update the Black Duck project and project version according to configured project and project version properties. (By default, these properties are only set on created projects / project versions.)

When set to true, the following properties will be updated on the Project: description (detect.project.description), tier (detect.project.tier), and project level adjustments (detect.project.level.adjustments). The following properties will also be updated on the project version: notes (detect.project.version.notes), phase (detect.project.version.phase), distribution (detect.project.version.distribution), nickname (detect.project.version.nickname), license (detect.project.version.license).

Details	
Added	4.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Version Name

```
--detect.project.version.name
```

An override for the version to use for the Black Duck project. If not supplied, Detect will attempt to use the tools to figure out a reasonable version name. If that fails, the current date will be used.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Version Nickname

```
--detect.project.version.nickname
```

If a project version nickname is specified, your project version will be created with this nickname. For updates, see detect.project.version.update.

Details	
Added	5.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Version Notes

```
--detect.project.version.notes
```

If project version notes are specified, your project version will be created with these notes. For updates, see detect.project.version.update.

Details	
Added	3.1.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Version Phase

```
--detect.project.version.phase=ARCHIVED,DEPRECATED,DEVELOPMENT,PLANNING
,PRERELEASE,RELEASED
```

If project version phase is specified, your project version will be created with this phase. For updates, see detect.project.version.update.

Details	
Added	3.0.0
Type	ProjectVersionPhaseType
Default Value	DEVELOPMENT
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	ARCHIVED, DEPRECATED, DEVELOPMENT, PLANNING, PRERELEASE, RELEASED
Strict	Yes

Allow Project Level Adjustments (Advanced)

```
--detect.project.level.adjustments=true
```

If set, created projects will be created with the value of this property. For updates, see detect.project.version.update.

Corresponds to the 'Always maintain component adjustments to all versions of this project' checkbox under 'Component Adjustments' on the Black Duck Project settings page.

Details	
Added	3.0.0
Type	Boolean
Default Value	true
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Application ID (Advanced)

```
--detect.project.application.id
```

Sets the 'Application ID' project setting.

Details	
Added	5.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Clone Latest Project Version (Advanced)

```
--detect.clone.project.version.latest=false
```

If set to true, detect will attempt to use the latest project version as the clone for this project. The project must exist and have at least one version.

Details	
Added	5.6.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Clone Project Categories (Advanced)

```
--detect.project.clone.categories=ALL,NONE,COMPONENT_DATA,CUSTOM_FIELD_
DATA,DEEP_LICENSE,LICENSE_TERM_FULFILLMENT,VERSION_SETTINGS,VULN_DATA
```

The value of this property is used to set the 'Cloning' settings on created Black Duck projects. If property detect.project.version.update is set to true, the value of this property is used to set the 'Cloning' settings on updated Black Duck projects.

Details	
Added	4.2.0
Type	ProjectCloneCategoriesType List
Default Value	ALL
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	ALL, NONE, COMPONENT_DATA, CUSTOM_FIELD_DATA, DEEP_LICENSE, LICENSE_TERM_FULFILLMENT, VERSION_SETTINGS, VULN_DATA
Strict	Yes

Clone Project Version Name (Advanced)

```
--detect.clone.project.version.name
```

The name of the project version to clone this project version from. Respects the given Clone Categories in detect.project.clone.categories or as set on the Black Duck server.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Custom Fields (Advanced)

```
--detect.custom.fields.project
```

A list of custom fields with a label and comma-separated value starting from index 0. For each index, provide one label and one value. For example, to set a custom field with label 'example' to 'one,two': `detect.custom.fields.project[0].label='example'` and `detect.custom.fields.project[0].value='one,two'`. To set another field, use index 1. Note that these will not show up in the detect configuration log.

When assigning a value that contains a comma to a single-value field such as a text field, append '[0]' to the end of the value property name. For example, to set the value of the first field you are setting ('detect.custom.fields.version[0]') to 'text1,text2', use 'detect.custom.fields.version[0].value[0]=text1,text2'.

Details	
Added	5.6.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Custom Fields (Advanced)

```
--detect.custom.fields.version
```

A list of custom fields with a label and comma-separated value starting from index 0. For each index, provide one label and one value. For example , to set a custom field with label 'example' to 'one,two': `detect.custom.fields.version[0].label='example'` and `detect.custom.fields.version[0].value='one,two'`. To set another field, use index 1. Note that these will not show up in the detect configuration log.

When assigning a value that contains a comma to a single-value field such as a text field, append '[0]' to the end of the value property name. For example, to set the value of the first field you are setting ('detect.custom.fields.version[0]') to 'text1,text2', use 'detect.custom.fields.version[0].value[0]=text1,text2'.

Details	
Added	5.6.0
Type	Optional String
Default Value	
Comma Separated	No

Details	
Case Sensitive	No
Acceptable Values	Any
Strict	No

Parent Project Name (Advanced)

```
--detect.parent.project.name
```

When a parent project and version name are specified, the created detect project will be added as a component to the specified parent project version. The specified parent project and parent project version must exist on Black Duck.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Parent Project Version Name (Advanced)

```
--detect.parent.project.version.name
```

When a parent project and version name are specified, the created detect project will be added as a component to the specified parent project version. The specified parent project and parent project version must exist on Black Duck.

Details	
Added	3.0.0
Type	Optional String

Details	
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Project Group Name (Advanced)

```
--detect.project.group.name
```

Sets the 'Project Group' to assign the project to. Must match exactly to an existing project group on Black Duck.

Details	
Added	7.8.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Project Tags (Advanced)

```
--detect.project.tags
```

A comma-separated list of tags to add to the project. This property is not supported when using Synopsys Detect in offline mode.

Details	
Added	5.6.0
Type	String List

Details	
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`Critical`

Project User Groups (Advanced)

```
--detect.project.user.groups
```

A comma-separated list of names of user groups to add to the project.

Details	
Added	5.4.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`ProjectManagers,TechLeads`

Scan Name (Advanced)

```
--detect.code.location.name
```

An override for the base name Detect will use for the scan (codelocation) it creates. Detect appends a suffix to the base name that indicates the source ("scan" for the signature scanner, "gradle/bom" for the Gradle detector, etc.). If this property is set and multiple code locations are generated from the same source, Detect will also append an index to avoid name collisions. When this property is set, detect.project.codelocation.prefix and detect.project.codelocation.suffix are ignored.

Details	
Added	4.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Scan Name Prefix (Advanced)

```
--detect.project.codelocation.prefix
```

A prefix to the name of the scans created by Detect. Useful for running against the same projects on multiple machines.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Scan Name Suffix (Advanced)

```
--detect.project.codelocation.suffix
```

A suffix to the name of the scans created by Detect.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Unmap All Other Scans for Project (Advanced)

```
--detect.project.codelocation.unmap=false
```

If set to true, unmaps all other scans mapped to the project version produced by the current run of Detect.

Details	
Added	4.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Version Distribution (Advanced)

```
--detect.project.version.distribution=EXTERNAL,INTERNAL,OPENSOURCE,SAAS
```

If project version distribution is specified, your project version will be created with this distribution. For updates, see detect.project.version.update.

Details	
Added	3.0.0
Type	ProjectVersionDistributionType
Default Value	EXTERNAL
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	EXTERNAL, INTERNAL, OPENSOURCE, SAAS
Strict	Yes

proxy

Bypass Proxy Hosts (Advanced)

```
--blackduck.proxy.ignored.hosts
```

A comma separated list of regular expression host patterns that should not use the proxy.

This property accepts Java regular expressions. Refer to the *Configuring Synopsys Detect* > *Java regular expression support* page for more details.

Details	
Added	4.2.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`blackduck[0-9]+.mycompany.com`

NTLM Proxy Domain (Advanced)

```
--blackduck.proxy.ntlm.domain
```

NTLM Proxy domain.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

NTLM Proxy Workstation (Advanced)

```
--blackduck.proxy.ntlm.workstation
```

NTLM Proxy workstation.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Proxy Host (Advanced)

```
--blackduck.proxy.host
```

Hostname of the proxy server.

Schema/protocol is not accepted by this parameter.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`--blackduck.proxy.host=<Proxy_IP/URL>`

Proxy Password (Advanced)

```
--blackduck.proxy.password
```

Proxy password.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Proxy Port (Advanced)

```
--blackduck.proxy.port
```

Proxy port number.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Proxy Username (Advanced)

```
--blackduck.proxy.username
```

Proxy username.

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

report

Generate Notices Report

```
--detect.notices.report=false
```

When set to true, a Black Duck notices report in text form will be created in your source directory.

Details	
Added	3.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Generate Risk Report (PDF)

```
--detect.risk.report.pdf=false
```

When set to true, a Black Duck risk report in PDF form will be created.

Details	
Added	3.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Notices Report Path

```
--detect.notices.report.path
```

The output directory for notices report. Default is the source directory.

Details	
Added	3.0.0

Details	
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Risk Report Output Path

```
--detect.risk.report.pdf.path
```

The output directory for risk report in PDF. Default is the source directory.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

signature-scanner

Detect Excluded Directories Search Depth

```
--detect.excluded.directories.search.depth=4
```

Enables you to adjust the depth to which Detect will search when creating signature scanner exclusion patterns.

Details	
Added	7.0.0
Type	Integer
Default Value	4
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Individual File Matching

```
--detect.blackduck.signature.scanner.individual.file.matching=NONE,SOUR
CE,BINARY,ALL
```

Users may set this property to indicate what types of files they want to match. Corresponding Signature Scanner CLI Argument: --individualFileMatching.

Details	
Added	6.2.0
Type	IndividualFileMatching
Default Value	NONE
Comma Separated	No
Case Sensitive	No
Acceptable Values	NONE, SOURCE, BINARY, ALL
Strict	Yes

Reduced Persistence

```
--detect.blackduck.signature.scanner.reduced.persistence=DEFAULT,RETAIN
_UNMATCHED,DISCARD_UNMATCHED
```

Use this value to control how unmatched files from signature scans are stored. For a full explanation, please refer to *about reduced persistence signature scanning.*

Details	
Added	8.3.0
Type	ReducedPersistence
Default Value	DEFAULT
Comma Separated	No
Case Sensitive	No
Acceptable Values	DEFAULT, RETAIN_UNMATCHED, DISCARD_UNMATCHED
Strict	Yes

Signature Scanner Arguments

```
--detect.blackduck.signature.scanner.arguments
```

A space-separated list of additional arguments to use when running the Black Duck signature scanner.

For example: Suppose you are running in bash on Linux and want to use the signature scanner's ability to read a list of directories to exclude from a file (using the signature scanner --exclude-from option). You tell the signature scanner read excluded directories from a file named excludes.txt in the current working directory with: --detect.blackduck.signature.scanner.arguments='--exclude-from ./excludes.txt'

Details	
Added	4.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`--exclude-from ./excludes.txt`

Signature Scanner Copyright Search

```
--detect.blackduck.signature.scanner.copyright.search=false
```

When set to true, user will be able to scan and discover copyright names in Black Duck. Corresponding Signature Scanner CLI Argument: --copyright-search.

Details	
Added	6.4.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Signature Scanner Dry Run

```
--detect.blackduck.signature.scanner.dry.run=false
```

If set to true, the signature scanner results are not uploaded to Black Duck, and the scanner results are written to disk via the Signature Scanner CLI argument: --dryRunWriteDir.

Details	
Added	4.2.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Signature Scanner License Search

```
--detect.blackduck.signature.scanner.license.search=false
```

When set to true, user will be able to scan and discover license names in Black Duck. Corresponding Signature Scanner CLI Argument: --license-search.

Details	
Added	6.2.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Signature Scanner Local Path

```
--detect.blackduck.signature.scanner.local.path
```

To use a local signature scanner, specify the path where the signature scanner was unzipped. This will likely look similar to 'scan.cli-x.y.z' and includes the 'bin, icon, jre, and lib' directories of the expanded scan.cli.

Details	
Added	4.2.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Signature Scanner Target Paths

```
--detect.blackduck.signature.scanner.paths
```

If this property is not set, the signature scanner target path is the source path (see property detect.source.path). If this property is set, the paths provided in this property's value will be signature scanned instead (the signature scanner will be executed once for each provided path).

Details	
Added	4.2.0
Type	Path List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No

Snippet Matching

```
--detect.blackduck.signature.scanner.snippet.matching=NONE,SNIPPET_MATC
HING,SNIPPET_MATCHING_ONLY,FULL_SNIPPET_MATCHING,FULL_SNIPPET_MATCHING_
ONLY
```

Use this value to enable the various snippet scanning modes. For a full explanation, please refer to the 'Running a component scan using the Signature Scanner command line' section in your Black Duck server's online help. Corresponding Signature Scanner CLI Arguments: --snippet-matching, --snippet-matching-only, --full-snippet-scan.

Details	
Added	5.5.0
Type	SnippetMatching
Default Value	NONE
Comma Separated	No

Details	
Case Sensitive	No
Acceptable Values	NONE, SNIPPET_MATCHING, SNIPPET_MATCHING_ONLY, FULL_SNIPPET_MATCHING, FULL_SNIPPET_MATCHING_ONLY
Strict	Yes

Upload source mode

```
--detect.blackduck.signature.scanner.upload.source.mode=false
```

If set to true, the signature scanner will, if supported by your Black Duck version, upload source code to Black Duck. Corresponding Signature Scanner CLI Argument: --upload-source.

Details	
Added	5.4.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Signature Scanner Memory (Advanced)

```
--detect.blackduck.signature.scanner.memory=4096
```

The memory for the scanner to use.

Details	
Added	4.2.0
Type	Integer

Details	
Default Value	4096
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Detector Properties

This section contains details about properties that configure the Synopsys Detect detectors.

Synopsys Detect uses detectors to find and extract dependencies from supported package managers.

By default, all detectors are eligible to run. The set of detectors that actually run depends on the files that exist in your project directory.

Note: Not all low accuracy detectors, see *Detector search and accuracy*, support configuration properties. See specific *Package Manager* pages for further information.

bazel

Bazel cquery additional options

```
--detect.bazel.cquery.options
```

A comma-separated list of additional options to pass to the bazel cquery command.

Details	
Added	6.1.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

Bazel Executable

```
--detect.bazel.path
```

The path to the Bazel executable.

Details	
Added	5.2.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Bazel Target

```
--detect.bazel.target
```

The Bazel target (for example, //foo:foolib) for which dependencies are collected. For Detect to run Bazel, this property must be set.

Details	
Added	5.2.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

Bazel workspace rules

```
--detect.bazel.workspace.rules=ALL,NONE,MAVEN_JAR,MAVEN_INSTALL,HASKELL
_CABAL_LIBRARY,HTTP_ARCHIVE
```

By default Detect discovers Bazel dependencies using all of the supported Bazel workspace rules that it finds in the WORKSPACE file. Alternatively you can use this property to specify the list of Bazel workspace rules Detect should use.

Setting this property (or letting it default) to NONE tells Detect to use supported rules that it finds in the WORKSPACE file.

Details	
Added	7.12.0
Type	WorkspaceRule List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	ALL, NONE, MAVEN_JAR, MAVEN_INSTALL, HASKELL_CABAL_LIBRARY, HTTP_ARCHIVE
Strict	Yes

bitbake

Bitbake Excluded Dependency Types

```
--detect.bitbake.dependency.types.excluded=NONE,BUILD
```

The dependency types to exclude from the results.

BUILD dependencies include recipes that are not declared in the license.manifest file, and native recipes. When excluding BUILD dependencies, Detect requires the license.manifest file (found under the {builddir}/tmp directory).

Details	
Added	7.10.0
Type	BitbakeDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, BUILD
Strict	Yes
Example	`BUILD`

BitBake Init Script Name

```
--detect.bitbake.build.env.name=oe-init-build-env
```

The name of the build environment init script.

Details	
Added	4.4.0
Type	String
Default Value	oe-init-build-env
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

BitBake Package Names

```
--detect.bitbake.package.names
```

A comma-separated list of package names from which dependencies are extracted.

Details	
Added	4.4.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No

BitBake Search Depth

```
--detect.bitbake.search.depth=1
```

The depth at which Detect will search for files generated by Bitbake.

Details	
Added	6.1.0
Type	Integer
Default Value	1
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

BitBake Source Arguments

```
--detect.bitbake.source.arguments
```

A comma-separated list of arguments to supply when sourcing the build environment init script.

Details	
Added	6.0.0

Details	
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No

conan

Additional Conan Arguments

```
--detect.conan.arguments
```

A space-separated list of additional arguments to add to the 'conan info' command line when running Detect against a Conan project. Detect will execute the command 'conan info {additional arguments} .'

Details	
Added	6.8.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`"--profile clang --profile cmake_316"`

Attempt Package Revision Match

```
--detect.conan.attempt.package.revision.match=false
```

If package revisions are available (a Conan lock file is found or provided, and Conan's revisions feature is enabled), require that each dependency's package revision match the package revision of the component in the KB.

Details	
Added	6.8.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Conan Dependency Types Excluded

```
--detect.conan.dependency.types.excluded=NONE,BUILD
```

Set this value to indicate which Conan dependency types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	ConanDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, BUILD
Strict	Yes
Example	`BUILD`

Conan Executable

```
--detect.conan.path
```

The path to the conan executable.

Details	
Added	6.8.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Conan Lockfile

```
--detect.conan.lockfile.path
```

The path to the conan lockfile to apply when running 'conan info' to get the dependency graph. If set, Detect will execute the command 'conan info --lockfile {lockfile} .'

Details	
Added	6.8.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

conda

Anaconda Environment Name

```
--detect.conda.environment.name
```

The name of the anaconda environment used by your project.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Conda Executable

```
--detect.conda.path
```

The path to the conda executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

cpan

cpan Executable

```
--detect.cpan.path
```

The path to the cpan executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

cpanm Executable

```
--detect.cpanm.path
```

The path to the cpanm executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

dart

dart Executable

```
--detect.dart.path
```

The path to the dart executable.

Details	
Added	7.5.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Dart Pub Dependency Types Excluded

```
--detect.pub.dependency.types.excluded=NONE,DEV
```

Set this value to indicate which Dart pub dependency types Detect should exclude from the BOM.

If DEV is excluded, the Dart Detector will pass the option --no-dev when running the command 'pub deps'.

Details	
Added	7.10.0
Type	DartPubDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, DEV
Strict	Yes
Example	`DEV`

flutter Executable

```
--detect.flutter.path
```

The path to the flutter executable.

Details	
Added	7.5.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

docker

Docker Executable

```
--detect.docker.path
```

Path to the docker executable (used to load image inspector Docker images in order to run the Docker Inspector in air gap mode).

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`/usr/local/bin/docker`

Docker Image ID

```
--detect.docker.image.id
```

The ID (shown in the 'IMAGE ID' column of 'docker images' output) of the target Docker image. The target image must already be local (must appear in the output of 'docker images').

detect.docker.image, detect.docker.tar, and detect.docker.image.id are three alternative ways to specify an image (you should only set one of these properties).

Details	
Added	6.1.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`fe1cc5b91830`

Docker Image Name

```
--detect.docker.image
```

The Docker image name (repo:tag) to inspect.

For Detect to run Docker Inspector, either this property, detect.docker.tar, or detect.docker.image.id must be set. Docker Inspector finds packages installed by the Linux package manager in Linux-based images. detect.docker.image, detect.docker.tar, and detect.docker.image.id are three alternative ways to specify an image (you should only set one of these properties). When a value of this property is provided, Docker Inspector will use the Docker engine to pull the image.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Details	
Example	`centos:centos8`

Image Archive File

```
--detect.docker.tar
```

An image .tar file which is either a Docker image saved to a file using the 'docker save' command, or an Open Container Initiative (OCI) image .tar file. The file must be readable by all.

detect.docker.image, detect.docker.tar, and detect.docker.image.id are three alternative ways to specify an image (you should only set one of these properties). The .tar file must conform to either of the following image format specifications: 1. Docker Image Specification v1.2.0 (https://github.com/moby/moby/blob/master/image/spec/v1.2.md), which is the format produced by the "docker save" command, or 2. Open Container Initiative Image Format Specification (https://github.com/opencontainers/image-spec/blob/main/spec.md).

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`./ubuntu21_04.tar`

Docker Inspector Path (Advanced)

```
--detect.docker.inspector.path
```

Use this property to point Detect to a local Docker Inspector jar file, instead of the default Docker Inspector jar file that Detect downloads from the binary repository. You need to ensure the version is compatible (the same major version that Detect downloads by default).

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Docker Inspector Version (Advanced)

```
--detect.docker.inspector.version
```

Version of the Docker Inspector to use. By default Detect will attempt to automatically determine the version to use.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`9.1.1`

Docker Passthrough (Advanced)

```
--detect.docker.passthrough
```

Additional properties may be passed to the docker inspector by adding the prefix detect.docker.passthrough to each Docker Inspector property name and assigning a value. The

'detect.docker.passthrough' prefix will be removed from the property name to generate the property name passed to Docker Inspector (with the given value).

Details	
Added	6.0.0
Type	None
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`(This example is unusual in that it shows a complete propertyname=value) detect.docker.passthrough.imageinspector.service.log.length=1000`

Platform Top Layer ID (Advanced)

```
--detect.docker.platform.top.layer.id
```

To exclude components from platform layers from the results, assign to this property the ID of the top layer of the platform image. Get the platform top layer ID from the output of 'docker inspect platformimage:tag'. The platform top layer ID is the last item in RootFS.Layers. For more information, see 'Isolating application components' in the Docker Inspector documentation.

If you are interested in components from the application layers of your image, but not interested in components from the underlying platform layers, you can exclude components from platform layers from the results by using this property to specify the boundary between platform layers and application layers.

Details	
Added	6.1.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No
Example	`sha256:f6253634dc78da2f2e3bee9c80635 93f880dc35d701307f30f65553e0f50c18c`

go

Go Executable

```
--detect.go.path
```

Path to the Go executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Go Mod Dependency Types Excluded

```
--detect.go.mod.dependency.types.excluded=NONE,UNUSED,VENDORED
```

Set this value to indicate which Go Mod dependency types Detect should exclude from the BOM.

If UNUSED is provided, Detect will use the results of 'go mod why' to filter out unused dependencies from Go modules declaring Go 1.16 or higher. If VENDORED is provided, Detect will use the results of 'go mod why -vendor' to filter out all unused dependencies.

Details	
Added	7.10.0
Type	GoModDependencyType

Details	
Default Value	NONE
Comma Separated	No
Case Sensitive	Yes
Acceptable Values	NONE, UNUSED, VENDORED
Strict	Yes
Example	`VENDORED`

gradle

Gradle Build Command

```
--detect.gradle.build.command
```

Gradle command line arguments to add to the gradle/gradlew command line.

By default, Detect runs the gradle (or gradlew) command with one task: dependencies. You can use this property to insert one or more additional gradle command line arguments (options or tasks) before the dependencies argument.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Gradle Configuration Types Excluded

```
--detect.gradle.configuration.types.excluded=NONE,UNRESOLVED
```

Set this value to indicate which Gradle configuration types Detect should exclude from the BOM.

Including dependencies from unresolved Gradle configurations could lead to false positives. Dependency versions from an unresolved configuration may differ from a resolved one. See https://docs.gradle.org/7.2/userguide/declaring_dependencies.html#sec:resolvable-consumable-configs

Details	
Added	7.10.0
Type	GradleConfigurationType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, UNRESOLVED
Strict	Yes
Example	`UNRESOLVED`

Gradle Executable

```
--detect.gradle.path
```

The path to the Gradle executable (gradle or gradlew).

If set, Detect will use the given Gradle executable instead of searching for one.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Gradle Exclude Configurations (Advanced)

```
--detect.gradle.excluded.configurations
```

A comma-separated list of Gradle configurations to exclude.

As Detect examines the Gradle project for dependencies, Detect will skip any Gradle configurations specified via this property. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Gradle Exclude Projects (Advanced)

```
--detect.gradle.excluded.projects
```

A comma-separated list of Gradle subprojects to exclude.

As Detect examines the Gradle project for dependencies, Detect will skip any Gradle subprojects specified via this property. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes

Details	
Acceptable Values	Any
Strict	No

Gradle Exclude Subproject Paths (Advanced)

```
--detect.gradle.excluded.project.paths
```

A comma-separated list of Gradle subproject paths to exclude.

As Detect examines the Gradle project for dependencies, Detect will skip any Gradle subproject whose path matches one of the values passed via this property. Run 'gradle projects' to see the paths to your subprojects. Subproject paths have the form ':subproject:subsubproject' and are unique. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	7.12.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Gradle Include Configurations (Advanced)

```
--detect.gradle.included.configurations
```

A comma-separated list of Gradle configurations to include.

As Detect examines the Gradle project for dependencies, if this property is set, Detect will include only those Gradle configurations specified via this property that are not excluded. Leaving this unset implies 'include all'. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Gradle Include Project Paths (Advanced)

```
--detect.gradle.included.project.paths
```

A comma-separated list of Gradle subproject paths to include.

As Detect examines the Gradle project for dependencies, if this property is set, Detect will include only those subprojects whose path matches this property. Gradle project paths usually take the form ':parent:child' and are unique. Leaving this unset implies 'include all'. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Details	
Added	7.12.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Gradle Include Projects (Advanced)

```
--detect.gradle.included.projects
```

A comma-separated list of Gradle subprojects to include.

As Detect examines the Gradle project for dependencies, if this property is set, Detect will include only those subprojects specified via this property that are not excluded. Leaving this unset implies 'include all'. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

hex

Rebar3 Executable

```
--detect.hex.rebar3.path
```

The path to the rebar3 executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

lerna

Lerna Executable

```
--detect.lerna.path
```

Path of the lerna executable.

Details	
Added	6.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Lerna Package Types Excluded

```
--detect.lerna.package.types.excluded=NONE,PRIVATE
```

Set this value to indicate which Lerna package types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	LernaPackageType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, PRIVATE
Strict	Yes
Example	`PRIVATE`

Lerna Packages Excluded (Advanced)

```
--detect.lerna.excluded.packages
```

A comma-separated list of Lerna packages to exclude.

As Detect parses the output of lerna ls --all --json, Detect will exclude any Lerna packages specified via this property. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Details	
Added	7.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Lerna Packages Included (Advanced)

```
--detect.lerna.included.packages
```

A comma-separated list of Lerna packages to include.

As Detect parses the output of lerna ls --all --json2, if this property is set, Detect will include only those Lerna packages specified via this property that are not excluded. Leaving this unset implies 'include all'. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Details	
Added	7.0.0
Type	String List
Default Value	
Comma Separated	Yes

Details	
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

maven

Dependency Scope Excluded

```
--detect.maven.excluded.scopes
```

A comma separated list of Maven scopes. Output will be limited to dependencies outside these scopes (overrides include).

If set, Detect will include only dependencies outside of the given Maven scope. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	6.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Dependency Scope Included

```
--detect.maven.included.scopes
```

A comma separated list of Maven scopes. Output will be limited to dependencies within these scopes (overridden by exclude).

If set, Detect will include only dependencies of the given Maven scope. This property accepts filename globbing-style wildcards. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	6.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

Maven Build Command

```
--detect.maven.build.command
```

Maven command line arguments to add to the mvn/mvnw command line.

By default, Detect runs the mvn (or mvnw) command with one argument: dependency:tree. You can use this property to insert one or more additional mvn command line arguments (goals, etc.) before the dependency:tree argument. For example: suppose you are running in bash on Linux, and want to point maven to your settings file (maven_dev_settings.xml in your home directory) and assign the value 'other' to property 'reason'. You could do this with: --detect.maven.build.command='--settings \${HOME}/maven_dev_settings.xml --define reason=other'

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Maven Executable

```
--detect.maven.path
```

The path to the Maven executable (mvn or mvnw).

If set, Detect will use the given Maven executable instead of searching for one.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Maven Modules Excluded (Advanced)

```
--detect.maven.excluded.modules
```

A comma-separated list of Maven modules (subprojects) to exclude.

As Detect parses the mvn dependency:tree output for dependencies, Detect will skip any Maven modules specified via this property. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any

Details	
Strict	No

Maven Modules Included (Advanced)

```
--detect.maven.included.modules
```

A comma-separated list of Maven modules (subprojects) to include.

As Detect parses the mvn dependency:tree output for dependencies, if this property is set, Detect will include only those Maven modules specified via this property that are not excluded. Leaving this unset implies 'include all'. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect > Property wildcard support* page for more details.

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No

npm

Additional NPM Command Arguments

```
--detect.npm.arguments
```

A space-separated list of additional arguments that Detect will add at then end of the npm ls command line when Detect executes the NPM CLI Detector on an NPM project.

Details	
Added	4.3.0
Type	Optional String

Details	
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`--depth=0`

Npm Dependency Types Excluded

```
--detect.npm.dependency.types.excluded=NONE,DEV,PEER
```

Set this value to indicate which Npm dependency types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	NpmDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, DEV, PEER
Strict	Yes
Example	`DEV,PEER`

NPM Executable

```
--detect.npm.path
```

The path to the Npm executable.

Details	
Added	3.0.0

Details	
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

nuget

Nuget Config File

```
--detect.nuget.config.path
```

The path to the Nuget.Config file to supply to the nuget exe.

Details	
Added	4.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Nuget Packages Repository URL

```
--detect.nuget.packages.repo.url=https://api.nuget.org/v3/index.json
```

The source for nuget packages

Set this to "https://www.nuget.org/api/v2/" if your are still using a nuget client expecting the v2 api.

Details	
Added	3.0.0
Type	String List
Default Value	https://api.nuget.org/v3/index.json
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any
Strict	No

Ignore Nuget Failures (Advanced)

```
--detect.nuget.ignore.failure=false
```

If true errors will be logged and then ignored.

Details	
Added	3.0.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Nuget Modules Included (Advanced)

```
--detect.nuget.included.modules
```

The names of the projects in a solution to include (overrides exclude). Detect will include all projects with names that include any of the given regex patterns. To match a full project name (for example: 'BaGet.Core'), use a regular expression that matches only the full name ('^BaGet.Core$')

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No
Example	`^BaGet.Core$,^BaGet.Core.Tests$`

Nuget Projects Excluded (Advanced)

```
--detect.nuget.excluded.modules
```

The projects within the solution to exclude. Detect will exclude all projects with names that include any of the given regex patterns. To match a full project name (for example: 'BaGet.Core'), use a regular expression that matches only the full name ('^BaGet.Core$')

Details	
Added	3.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No
Example	`^BaGet.Core$,^BaGet.Core.Tests$`

packagist

Packagist Dependency Types Excluded

```
--detect.packagist.dependency.types.excluded=NONE,DEV
```

Set this value to indicate which Packagist dependency types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	PackagistDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, DEV
Strict	Yes

pear

Pear Dependency Types Excluded

```
--detect.pear.dependency.types.excluded=NONE,OPTIONAL
```

Set this value to indicate which Pear dependency types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	PearDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, OPTIONAL
Strict	Yes
Example	`OPTIONAL`

Pear Executable

```
--detect.pear.path
```

The path to the pear executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

pip

Pipenv Executable

```
--detect.pipenv.path
```

The path to the Pipenv executable.

Details	
Added	4.1.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Pip Executable

```
--detect.pip.path
```

The path to the Pip executable.

Details	
Added	6.8.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

Pipfile Dependency Types Excluded

```
--detect.pipfile.dependency.types.excluded=NONE,DEV
```

A comma-separated list of dependency types that will be excluded.

If DEV is excluded, the Pipfile Lock Detector will exclude 'develop' dependencies when parsing the Pipfile.lock file.

Details	
Added	7.13.0
Type	PipenvDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, DEV
Strict	Yes
Example	`DEV`

PIP Include Only Project Tree

```
--detect.pip.only.project.tree=false
```

By default, pipenv includes all dependencies found in the graph. Set to true to only include dependencies found underneath the dependency that matches the provided pip project and version name.

Details	
Added	6.1.0
Type	Boolean
Default Value	false
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

PIP Project Name

```
--detect.pip.project.name
```

The name of your PIP project, to be used if your project's name cannot be correctly inferred from its setup.py file.

Details	
Added	3.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

PIP Project Version Name

```
--detect.pip.project.version.name
```

The version of your PIP project, to be used if your project's version name cannot be correctly inferred from its setup.py file.

Details	
Added	4.1.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

PIP Requirements Path

```
--detect.pip.requirements.path
```

A comma-separated list of paths to requirements files, to be used to analyze requirements files with a filename other than requirements.txt or to specify which requirements files should be analyzed.

This property should only be set if you want the PIP Inspector Detector to run. For example: If your project uses Pipenv, do not set this property.

Details	
Added	3.0.0
Type	Path List
Default Value	
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	Any

Details	
Strict	No

python

Python Executable

```
--detect.python.path
```

The path to the Python executable.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No

ruby

Ruby Dependency Types Excluded

```
--detect.ruby.dependency.types.excluded=NONE,RUNTIME,DEV
```

Set this value to indicate which Ruby(Gempsec) dependency types Detect should exclude from the BOM.

Details	
Added	7.10.0
Type	GemspecDependencyType List
Default Value	NONE
Comma Separated	Yes

Details	
Case Sensitive	No
Acceptable Values	NONE, RUNTIME, DEV
Strict	Yes
Example	`DEV,RUNTIME`

sbt

Additional sbt command Arguments

```
--detect.sbt.arguments
```

A space-separated list of additional arguments to add to sbt command line when running Detect against an SBT project. Detect will execute the command 'sbt {additional arguments} {Detect-added arguments}'.

Details	
Added	7.0.0
Type	Optional String
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`"-Djline.terminal=jline.UnsupportedTerminal"`

Sbt Executable

```
--detect.sbt.path
```

Path to the Sbt executable.

If set, Detect will use the given Sbt executable instead of searching for one.

Details	
Added	3.0.0
Type	Optional Path
Default Value	
Comma Separated	No
Case Sensitive	No
Acceptable Values	Any
Strict	No
Example	`C:\Program Files (x86)\sbt\bin\sbt.bat`

yarn

Yarn Dependency Types Excluded

```
--detect.yarn.dependency.types.excluded=NONE,NON_PRODUCTION
```

Set this value to indicate which Yarn dependency types Detect should exclude from the BOM.

Details	
Added	4.0.0
Type	YarnDependencyType List
Default Value	NONE
Comma Separated	Yes
Case Sensitive	No
Acceptable Values	NONE, NON_PRODUCTION
Strict	Yes
Example	`NON_PRODUCTION`

Yarn Exclude Workspaces (Advanced)

```
--detect.yarn.excluded.workspaces
```

A comma-separated list of Yarn workspaces (specified by the workspace directory's relative path) to exclude.

By default, Detect includes all workspaces, but will skip any Yarn workspaces specified via this property. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	7.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any
Strict	No
Example	`workspaces/workspace-a,workspaces/*-test`

Yarn Include Workspaces (Advanced)

```
--detect.yarn.included.workspaces
```

A comma-separated list of Yarn workspaces (specified by the workspace directory's relative path) to include.

By default, Detect includes all workspaces. If workspaces are excluded or included, Detect will include any workspace included by this property that is not excluded. Exclusion rules always win. This property accepts filename globbing-style wildcards. Refer to the *Configuring Synopsys Detect* > *Property wildcard support* page for more details.

Details	
Added	7.0.0
Type	String List
Default Value	
Comma Separated	Yes
Case Sensitive	Yes
Acceptable Values	Any

Details	
Strict	No
Example	`workspaces/workspace-a,workspaces/workspace-b`

Deprecated Properties

This page lists Synopsys Detect's deprecated properties; for both active and deprecated properties, refer to *all properties*.

debug

Property	Description
detect.diagnostic.extended	default: false Diagnostic Mode Extended: When enabled, Synopsys Detect performs the actions of --detect.diagnostic, but also includes relevant files such as lock files and build artifacts. **DEPRECATED: This property is being removed. Use property detect.diagnostic instead. There is no longer any distinction between extended and non-extended diagnostic zip files. This property will be removed in 9.0.0.**

12

Viewing and Managing Scan Results

Online mode

To view and manage your Synopsys Detect scan results after running Synopsys Detect online, do the following.

- In the Synopsys Detect output look for "Detect Result" and copy the Black Duck Project BOM URL as shown in the following example:

```
2020-06-11 06:35:39 INFO [main] ---------- Detect Result ========
2020-06-11 06:35:39 INFO [main] --- Black Duck Project BOM: https://my-hub-docker/api/projects/d8f798f1-1901-4902-aec7-f2e1cf2e4958/versions/6a8938e9-3615-40dd-8386-3bcb4ba52bec/components
```

- Open the Black Duck Project BOM URL in a browser to view the scan results in Black Duck.
- To find your scan in Black Duck, go to your Black Duck instance and click Scans to see a list of scans on the Scans page.

For help with viewing and analyzing your scan results go to the Black Duck Help page navigation menu at https://<Your hub host>/doc/Welcome.htm

Offline mode

To view and manage your Synopsys Detect scan results after running Synopsys Detect offline (with property *blackduck.offline.mode* set to *true*), do the following.

- In the Synopsys Detect output (near the beginning), look for the value of "Run directory". The output files will be written into subdirectories of the run directory. For example:

```
2022-03-07 15:46:29 EST INFO  [main] --- Run directory: /Users/billings/blackduck/runs/2022-03-07-20-46-29-611
```

Upload each of the output files (.bdio and .bdmu files, found in subdirectories of the run directory) into Black Duck using the Black Duck Scans page.

Risk Report Generation

Synopsys Detect can generate a Black Duck risk report in PDF form. Synopsys Detect looks for risk report generation details in the properties whose names start with detect.risk.report, including:

- detect.risk.report.pdf (enable report generation)
- detect.risk.report.path (path where report will be located)

Fonts

Default font files are used to create the risk report pdf.

You may specify a custom regular font and/or a custom bold font by placing a .ttf font file in a directory called "custom-regular" and/or "custom-bold", respectively, that is a child to the directory at detect-output-directory/tools/fonts, where 'detect-output-directory' is determined by *detect.output.path*

Examples

- /path-I-passed-to-detect-output-path/tools/fonts/custom-regular/my-custom-regular-font.ttf
- /Users/user/blackduck/tools/fonts/custom-regular/my-custom-regular-font.ttf on Unix
- C:\Users\blackduck\tools\fonts\custom-bold\my-custom-bold-font.ttf on Windows

Air Gap

Normally font files used in creating the risk report pdf are downloaded from Artifactory. If you are using the Synopsys Detect air gap, the font files are retrieved from a directory called 'fonts' that is a child to the root of the air gap directory.

To specify custom fonts when using the Synopsys Detect air gap zip, you must unzip the produced airgap zip file and then place a .ttf font file in a directory called "custom-regular" and/or "custom-bold" that is a child to the directory airGapRoot/fonts.

Example

- synopsys-detect-detect_version-air-gap/fonts/custom-regular/my-custom-regular-font.ttf

13

Troubleshooting

This section contains guidance related to troubleshooting Synopsys Detect issues.

Getting information

Simple issues

- Run Synopsys Detect with `--logging.level.detect=DEBUG` (the default logging level, INFO, is insufficient for troubleshooting) and read through the entire log for clues.
- For additional detail, run with `--logging.level.detect=TRACE`.
- Synopsys Detect typically runs package manager commands or build tool commands similar to commands used in your build. When run by Synopsys Detect, those commands (as well as the environment in which they run) need to be consistent with your build, and it's important to verify that they are. For example, the Gradle detector defaults to running *./gradlew dependencies* if it finds the file ./gradlew. If your build runs a different Gradle command or wrapper such as /usr/local/bin/gradle, use property *detect.gradle.path* to tell Synopsys Detect to run the same Gradle command that your build runs. Check the DEBUG log for the package manager commands that Synopsys Detect is running, and compare them to the commands your build runs.

More complex issues

For more complex issues, including any issue that requires help from Synopsys Technical Support, refer to *Diagnostic mode*.

Common problems

- See if you can reproduce the problem using the latest version of Synopsys Detect with the latest version of Black Duck. If not, the problem may be either fixed, or due to incompatible Synopsys Detect / Black Duck versions.
- Remember to consider the possibility that the Black Duck user lacks the necessary permissions (to create the project, update the BOM, receive notifications, etc.) in Black Duck. For more information, see *Black Duck user role requirements*.
- Remember to consider the possibility that the Black Duck server (registration key) may not have required capabilities enabled (binary upload, snippet scanning, etc.).

Incorrect or missing components

- For issues related to tools invoked by Synopsys Detect (Black Duck Signature Scanner, Docker Inspector, etc.), please check that tool's documentation.
- For issues related to incorrect components in the Black Duck BOM: Synopsys Detect has a great deal of control over matches produced by detectors (that are written to BDIO files), but no control over matches produced by the Black Duck Signature Scanner / Black Duck. When investigating an incorrect component in a Black Duck BOM, you need to determine whether the component was contributed by a detector, or by the Black Duck Signature Scanner: On the Black Duck Components tab for the project/version: Click on the "N Matches" link next to the component. The next screen lists the matches on the right side. Matches from the Black Duck Signature Scanner have a filename in the Name column. Matches from detectors have an external ID (such as "org.hamcrest:hamcrest-core:1.3") in the Name column.
- For issues related to components missing from or or incorrectly categorized in the Black Duck BOM: Synopsys Detect has a great deal of control over the production of BDIO files (use *--detect.diagnostic* to save these), but no control over how they are converted into a BOM by Black Duck. A good first step is to determine whether the BDIO files produced are correct. If they are incorrect, the problem is related to what Synopsys Detect is doing. If they are correct, but the BOM is incorrect, the problem is related to what Black Duck is doing. Similarly, Synopsys Detect is responsible for passing the correct arguments to the Black Duck Signature Scanner, but beyond that has no control over the results it produces.

Spring Boot related issues

- Synopsys Detect is a Spring Boot application, and leverages Spring Boot to provide various mechanism to configure it through *property settings*. This flexibility comes with a risk: it's possible for Synopsys Detect to be influenced by files (application.properties, application.xml, etc.) that may exist in the directory from which Synopsys Detect is run that are intended for some other application. This can produce some strange results. If properties have unexpected values (see the Synopsys Detect log), this is a possibility worth considering. The best solution may be simply to run Synopsys Detect from a different (ideally empty) directory (use the --detect.source.path argument).
- Similarly, Synopsys Detect can be influenced by environment variables via the same Spring Boot mechanism, so it's worth checking the environment for variables that correspond to Synopsys Detect property names.

Diagnostic mode

Synopsys Detect's diagnostic mode produces a diagnostic zip containing files needed for troubleshooting. If you contact Synopsys Technical Support for help troubleshooting an issue, you will need to provide a diagnostic zip from a Synopsys Detect run that exhibits the problem.

To enable diagnostic mode, add any one of the following to the command line:

- -d
- --diagnostic
- *--detect.diagnostic*

A diagnostic zip includes many files valuable for troubleshooting, such as:

- INFO, DEBUG, and TRACE level logs.
- The BDIO file produced by the run.
- Intermediary files generated by Synopsys Detect, such as Gradle inspector output files.
- Various reports.
- Relevant package manager files.
- The output of commands executed by Synopsys Detect (package manager commands, etc.).
- Logs and output files from other executed tools (Black Duck Signature Scanner, etc.).

After running Synopsys Detect in diagnostic mode, log messages similar to the following provide the path to the diagnostic zip file.

```
2019-03-29 09:32:02 INFO  [main] --- Creating diagnostics zip.
2019-03-29 09:32:02 INFO  [main] --- Diagnostics zip location: C:\detec
t\blackduck\runs\detect-run-2019-03-29-13-31-50-492.zip
2019-03-29 09:32:03 INFO  [main] --- Diagnostics file created at: C:det
ect\blackduck\runs\detect-run-2019-03-29-13-31-50-492.zip
2019-03-29 09:32:03 INFO  [main] --- Diagnostic mode has completed.
```

To conserve disk space, be sure to disable diagnostic mode once it is no longer needed.

Detect Exit Codes

Exit Code Key	Value	Description
SUCCESS	0	Detect exited successfully.
FAILURE_BLACKDUCK_CONNECTIVITY	1	Detect was unable to connect to Black Duck. Check your configuration and connection.

Exit Code Key	Value	Description
FAILURE_TIMEOUT	2	Detect was unable to wait for actions to be completed on Black Duck. Check your Black Duck server or increase your timeout.
FAILURE_POLICY_VIOLATION	3	Detect found policy violations.
FAILURE_PROXY_CONNECTIVITY	4	Detect was unable to use the configured proxy. Check your configuration and connection.
FAILURE_DETECTOR	5	Detect had one or more detector failures while extracting dependencies. Check that all projects build and your environment is configured correctly.
FAILURE_SCAN	6	Detect was unable to run the signature scanner against your source. Check your configuration.
FAILURE_CONFIGURATION	7	Detect was unable to start due to issues with it's configuration. Check and fix your configuration.
FAILURE_DETECTOR_REQUIRED	9	Detect did not run all of the required detectors. Fix detector issues or disable required detectors.
FAILURE_BLACKDUCK_VERSION_NOT_SUPPORTED	10	Detect's configuration requires a Black Duck capability that is not supported by your version of Black Duck. Ensure that your Black Duck version is compatible with this version of Detect.

Exit Code Key	Value	Description
FAILURE_BLACKDUCK_FEATURE_ERROR	11	Detect encountered an error while attempting an operation on Black Duck. Ensure that your Black Duck version is compatible with this version of Detect, and that your Black Duck user account has the required roles.
FAILURE_MINIMUM_INTERVAL_NOT_MET	13	Detect did not wait the minimum required scan interval.
FAILURE_IAC	14	Detect was unable to perform IaC Scan against your source. Please check your configuration, and see logs and IaC Scanner documentation for more information.
FAILURE_ACCURACY_NOT_MET	15	Detect was unable to meet the required accuracy.
FAILURE_IMAGE_NOT_AVAILABLE	20	Image scan attempted but no return data available.
FAILURE_GENERAL_ERROR	99	Detect encountered a known error, details of the error are provided.
FAILURE_UNKNOWN_ERROR	100	Detect encountered an unknown error.

Solutions to common problems

DETECT_SOURCE was not set or computed correctly

Symptom

detect8.sh fails with: DETECT_SOURCE was not set or computed correctly, please check your configuration and environment.

Possible cause

detect8.sh is trying to execute this command:

```
curl --silent --header \"X-Result-Detail: info\" https://sig-repo.synop
sys.com/api/storage/bds-integrations-release/com/synopsys/integration/s
ynopsys-detect?properties=DETECT_LATEST
```

The response to this command should be similar to the following:

```
{
"properties" : {
"DETECT_LATEST" : [ "https://sig-repo.synopsys.com/bds-integrations-rel
ease/com/synopsys/integration/synopsys-detect/5.6.1/synopsys-detect-5.6
.1.jar" ]
},
"uri" : "https://sig-repo.synopsys.com/api/storage/bds-integrations-rel
ease/com/synopsys/integration/synopsys-detect"
}
```

When that command does not successfully return a value for property DETECT_LATEST, detect8.sh reports:

```
DETECT_SOURCE was not set or computed correctly, please check your conf
iguration and environment.
```

Solution

If the curl command described above does not successfully return a value for property DETECT_LATEST, you must determine why, and make the changes necessary so that curl command works.

Synopsys Detect succeeds, but the results are incomplete because package managers or subprojects were overlooked

Symptom

In this scenario, everything succeeds, but many or all components are missed. Examining the log shows that package managers were not recognized and/or subprojects were overlooked.

Possible cause

The detector search depth needs to be increased. The default value (0) limits the search for package manager files to the project directory. If project manager files are located in subdirectories and/or there are subprojects, this depth should be increased to enable Synopsys Detect to find the relevant files, so it will run the appropriate detector(s).

See *detector search depth* for more details.

Synopsys Detect fails and a TRACE log shows an HTTP response from Black Duck of "402 Payment Required" or "502 Bad Gateway"

Symptom

Synopsys Detect fails, and a TRACE log contains "402 Payment Required" or "502 Bad Gateway".

Possible cause

Black Duck does not have a required feature (notifications, binary analysis, etc.) enabled.

Solution

Enable the required feature on the Black Duck server.

Unexpected behavior running Synopsys Detect on a project that uses Spring Boot

Symptom

Unexpected behavior, and/or unexpected property values shown in the log.

Possible cause

If your source directory contains Spring Framework configuration files named application.properties, application.yml, or application.xml that are written for any application other than Synopsys Detect, you should not run Synopsys Detect from your source directory.

Solution

To prevent Synopsys Detect from reading those files, run Synopsys Detect from a different directory. Use the following property to point to your source directory.

```
--detect.source.path={project directory path}
```

PKIX error connecting to Black Duck

Symptom

Exception: Could not communicate with Black Duck: Could not perform the authorization request: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target

Possible cause

The Black Duck server certificate is not in Java's keystore.

Solution

1. Acquire the certificate file for your Black Duck server.
2. Determine which *java* executable is being used to run Synopsys Detect. If you run detect8.sh, that is either $JAVA_HOME/bin/java (the default) or the first *java* found on your $PATH.
3. Determine the Java home directory for that *java* executable.
4. Run *keytool* to install the Black Duck server certificate into the keystore in that Java home directory.

Although not recommended, it is possible to disable the certificate check with the *trust cert property*.

Not Extractable: NUGET - Solution INFO [main] -- Exception occurred: java.nio.file.InvalidPathException

Symptom

Running Synopsys Detect on a NuGet project on Windows, a message similar to the following appears in the Synopsys Detect log:

```
Not Extractable: NUGET - Solution INFO [main] -- Exception occurred: ja
va.nio.file.InvalidPathException: Illegal char
<:> at index 2: C:\...
```

Possible cause

The value of $PATH contains a whitespace character after a semicolon and the path mentioned in the log message.

Solution

Remove spaces immediately following semicolons in the value of $PATH.

No project name/version provided or derived

Symptom

Upload to Black Duck fails with a message similar to the following in the log:

```
ERROR [main] -- createProject.arg0.name can't be blank [HTTP Error]: Th
ere was a problem trying to POST https://.../api/projects, response was
 412 Precondition Failed.
```

Possible cause

No project name and version were provided via properties and no Synopsys Detect tool capable of deriving a project name and version was included in the run. For example, you will get this (or a similar) error if you run with --detect.tools.BINARY_SCANNER and do not set --detect.project.name or --detect.project.version.name.

Solution

Set --detect.project.name and --detect.project.version.name.

Black Duck Signature Scanner fails on Alpine Linux

Symptom

The Black Duck Signature Scanner fails on Alpine Linux with an error similar to:

```
There was a problem scanning target '/opt/projects/myproject': Cannot r
un program "/home/me/blackduck/tools/Black_Duck_Scan_Installation/scan.
cli-2020.6.0/jre/bin/java": error=2, No such file or directory
```

Possible cause

The Java bundled with the Black Duck Signature Scanner does not work on Alpine Linux (it relies on libraries not usually present on an Alpine system).

Solution

Install an appropriate version of Java and tell Synopsys Detect to invoke the Black Duck Signature Scanner using that version of Java by setting environment variable BDS_JAVA_HOME to the JAVA_HOME value for that Java installation.

For example:

```
export BDS_JAVA_HOME=$JAVA_HOME
```

Or:

```
export BDS_JAVA_HOME=/usr/lib/jvm/java-11-openjdk/jre
```

On Windows: Error trying cleanup

Symptom

When running on Windows, inspecting a Docker image (e.g. using --detect.docker.image or --detect.docker.tar), during shutdown, Synopsys Detect logs messages similar to the following:

```
2020-08-14 14:31:04 DEBUG [main] --- Error trying cleanup:

java.io.IOException: Unable to delete file: C:\Users\Administrator\blac
```

```
kduck\runs\2020-08-14-21-28-40-106\extractions
...
Caused by: java.nio.file.FileSystemException: C:\Users\Administrator\bl
ackduck\runs\2020-08-14-21-28-40-106\extractions\DOCKER-0\application.p
roperties: The process cannot access the file because it is being used
by another process.
```

Possible cause

This happens when Docker fails to release its lock on the volume mounted directory when it shuts down the image inspector service container due to *Docker for Windows issue 394*. The result is that Synopsys Detect cannot fully clean up its output directory and leaves behind empty subdirectories. The problem may be intermittent.

Solution

There is no harm in leaving the directories behind in the short term but we recommend periodically removing them if the problem occurs frequently. Restarting Docker will force Docker to release the locks, and enable you to remove the directories.

Usage metrics collection

Synopsys Detect uses Google Analytics to collect anonymized usage metrics through a mechanism called *phone home*. Synopsys uses this data to help set engineering priorities.

In a network where access to outside servers is limited, this mechanism may fail, and those failures may be visible in the log. This is a harmless failure; Synopsys Detect will continue to function normally.

To disable this mechanism for Synopsys Detect runs executed from one environment, set the environment variable *SYNOPSYS_SKIP_PHONE_HOME* to *true*. To disable this mechanism for all Synopsys Detect runs against a specific Black Duck server, refer to the Black Duck documentation for information on disabling analytics.

14

Synopsys Detect Integrations

The Synopsys Detect integrations allow users to install and run Synopsys Detect in various environments.

Synopsys Detect scans code bases in your projects and folders to perform compositional analysis and functions as a Black Duck intelligent scan client. Synopsys Detect sends scan results to Black Duck, which generates risk analysis when identifying open source components, licenses, and security vulnerabilities.

Note: Plugins are independant open source projects, not all will be supported for a particular release of Synopsys Detect at the time of Synopsys Detect release. Please consult the *compatibility matrix* to determine the compatible releases.

Jenkins Plugin

The Synopsys Detect for Jenkins plugin enables you to install and run Synopsys Detect in your Jenkins instance.

Synopsys Detect scans code bases in your projects and folders to perform compositional analysis and functions as a Black Duck intelligent scan client. Synopsys Detect sends scan results to Black Duck, which generates risk analysis when identifying open source components, licenses, and security vulnerabilities.

Synopsys Detect is designed to run in the native build environment of the project that you want to scan. It uses the same global configuration as your Jenkins instance and provides a pass-through for Synopsys Detect. You can run as a post-build action in a Jenkins Freestyle job or run as a Pipeline step using a Pipeline script in a PipeLine job. After running a Synopsys Detect scan following the Jenkins build, you can view the scan results in your Black Duck instance.

Refer to *How it Works* to learn more about Synopsys Detect.

Basic workflow

1. Make sure you satisfy system and other requirements.

- Install the Synopsys Detect plugin in Jenkins.
- Configure Black Duck connection and plugin.

2. Run a Jenkins build on your project.

3. Synopsys Detect scans the project, for example, the scan might be a step in a Jenkins Pipeline job or post-build action in a Freestyle job.

4. Synopsys Detect sends the scan results to Black Duck for analysis.

After running a Synopsys Detect scan following the Jenkins build, you can view the scan results in your Black Duck instance.

Release Notes for Jenkins Plugin

Version 8.0.1

Resolved issues

- (IDTCTJNKNS-266) Resolved the following issues:
  - Jenkins 2.410 fails to start/crashes during startup if using the 8.0.0 plugin. *JENKINS-71480*
  - Inclusion of too many dependencies in version 8.0.0 *JENKINS-70671*
  - 8.0.0 bundles pull-parser.jar by mistake. *JENKINS-71023*

Important: Customers running Jenkins version 2.410 and above should upgrade to Synopsys Detect Jenkins plugin 8.0.1

Version 8.0.0

New features

- Updated to be compatible with Synopsys Detect 8.x.x. (Downloading and using detect8.(sh/ps1)).

Changed features

- The Jenkins plugin has been upgraded to use Synopsys Detect 8.x.x for execution.
- The plugin has been built against upgraded Jenkins/Jenkins plugin versions in order to mitigate known security risks.
- The minimal Jenkins version required is 2.377.
- Configuration and usage of the plugin is unchanged.

Resolved issues

- (IDTCTJNKNS-258) CVE-2022-42889 for Synopsys Detect Jenkins plugin 7.0.0
- (IDTCTJNKNS-261) Synopsys Detect v8 for Jenkins plugin
- (IDTCTJNKNS-255) Update dependency for Jenkins version, including optional plugin dependencies

- (IDTCTJNKNS-254) Only escape Synopsys Detect parameter values
- (IDTCTJNKNS-253) Improve clarity of messages logged when running plugin
- (IDTCTJNKNS-252) Update internal dependencies to latest
- (IDTCTJNKNS-247) Detect shell scripts are executed first and then downloaded in Pipeline execution in Linux and Windows slave nodes
- (IDTCTJNKNS-239) Avoid leaking API token string in the console output
- (IDTCTJNKNS-228) Unable to use java version specified in pipeline when running Synopsys Detect in Air Gap mode
- (IDTCTJNKNS-224) Improve clarity in the transition between the different stages of Synopsys Detect for Jenkins
- (IDTCTJNKNS-220) Jenkins Build is changed to Unstable for Invalid values in Synopsys Detect Installers
- (IDTCTJNKNS-192) Size must be between 1 and 50 when --detect.project.tag is more than 50 characters

Version 7.0.0

New features

- Update major version to match major version of Synopsys Detect that it runs.
- Update plugin to be compatible with Synopsys Detect 7.x.x. (Downloading and using detect7.(sh/ps1)).
  - Use property detect.timeout instead of blackduck.timeout
  - Remove support for using blackduck.password and blackduck.username and exclusively use blackduck.api.token
- Update UI when configuring plugin so that it will only list 'Secret Text' saved entries (Manage Jenkins -> Configure System -> Synopsys Detect -> Black Duck credentials)

Changed features

- When using script (sh/ps1), no longer cache the script. Plugin will download the script on each execution.

Version 3.1.0

New features

- Added the capability to run Synopsys Detect in Air Gap mode using the Synopsys Detect plugin.

Version 3.0.0

New features

- Added capability to turn off automatic escaping by setting the environment variable DETECT_PLUGIN_ESCAPING to false.

Resolved issues

- (IDTCTJNKNS-181) Resolved an issue wherein proxy details could not be determined from Jenkins when running a Black Duck job on a Jenkins agent because it only worked on the Jenkins main node.

Changed features

- The Polaris fields in the plugin are removed.
  - This functionality has moved to Synopsys Polaris for Jenkins.
- Updated the minimum version for Jenkins to 2.150.3.
- Connection validation is improved when testing through a proxy.

Version 2.1.1

Resolved issues

- Resolved an issue wherein Synopsys Detect for Jenkins didn't escape commas correctly in PowerShell arguments.
- Resolved an issue wherein Synopsys Detect for Jenkins didn't function when there were spaces in the workspace path resulting in failure to find the shell/PowerShell script.
- Version 2.0.2 of the SSynopsys Detect for Jenkins plugin violated semantic versioning by introducing a non-backward compatible change. Updating to any 2.X version from version 2.0.1 or earlier must be done with caution as that update might break existing functionality.

Version 2.1.0

New features

- Synopsys Detect for Jenkins now returns an exit code of 0 for a successful pipeline run.

Changed features

- On build failures, Synopsys Detect for Jenkins no longer modifies the build status when run in a Jenkins pipeline. Now, it throws an exception error if Detect fails.
- Synopsys Detect for Jenkins is improved to support the pipeline step context. Using *withEnv* and running Docker now works as expected.
- Added improvements for working with containers.
- Verified support for Synopsys Detect Jenkins plugin in the Cloudbees Core environment built with Kubernetes.

Version 2.0.2

New features

- Added auto-escaping parameters.

Changed features

- Now uses Synopsys Detect site to resolve the shell scripts.

Resolved issues

- Resolved an issue wherein a null pointer exception may be thrown when the proxy user name is blank.
- Resolved in issue wherein the plugin was not properly escaping the path to the PowerShell script. This also improves handling of elements like random pipes in the path.

Version 2.0.1

Resolved issues

- Resolved an issue wherein configuration/connection settings for the plugin are deleted when restarting Jenkins.

Version 2.0.0

New features

- You can now run Synopsys Detect for Jenkins by uploading a Detect JAR file.
- Synopsys Detect for Jenkins now uses the Polaris credentials stored in the credentials plugin in Jenkins.

Version 1.5.0

Resolved issues

- Resolved an issue wherein the proxy settings may be ignored when downloading the jar file.
- Resolved an issue that may have caused an error when connecting to the test repository.

Changed features

- Synopsys Detect for Jenkins now displays in parenthesis the version of Detect packaged with the plugin.

Version 1.4.1

- Maintenance release with overall improvements in stability and security.

Version 1.4.0

- Added support for converting from a Maven project to a Gradle project.
- Improved error handling for Synopsys Detect exit codes.
- Addressed an issue wherein cancelling a Synopsys Detect job was not terminating correctly.

Version 1.3.0

- Added support for Java 8.
- Synopsys Detect for Jenkins now supports Jenkins version 2.60.1 and higher.
- Added API key support.
- Added support for Microsoft NT Lan Manager (NTLM) protocol.

Version 1.2.0

- Added support for Java 7.
- Now includes support for an Artifactory URL override option.

Version 1.1.0

- Added DSL support.

Version 1.0.2

Resolved Issues

- Subordinate nodes do not use the proxy to download the Synopsys Detect *.jar* file (potential fix)

Version 1.0.1

Resolved Issues

- Resolved an issue wherein *JenkinsProxyHelper.shouldUseProxy* (final URL, final String noProxyHosts) was incorrectly returning *false* if the Hub URL was set, and incorrectly returning *true* when the Hub host name should be ignored.
- Resolved an issue with the Java executable path.

Version 1.0.0

- First release of product

Requirements for Synopsys Detect for Jenkins

Requirements for Synopsys Detect for Jenkins which may be in addition to those for Synopsys Detect.

- Access to the internet is required to download components from GitHub and other locations.
- Minimum 8GB RAM.
- Java: OpenJDK 64-bit version 8
- Jenkins minimum version of 2.377.

Downloading, Installing, and Updating the Plugin

To install the Synopsys Detect for Jenkins plugin, perform the following steps:

1. Navigate to **Manage Jenkins** > **Manage Plugins**.

2. Select the **Available** tab. (Note that if the plugin is already installed, it does not appear in the **Available** list.)
3. Select **Synopsys Detect**.
4. Click **Download now and install after restart**. This is the Synopsys recommendation for installing the plugin.
5. After restarting Jenkins, confirm that the plugin is successfully installed by navigating to **Manage Jenkins** > **Manage Plugins > Installed**, and verify that **Synopsys Detect** displays in the list.

Synopsys Detect plugin for Jenkins GitHub page *jenkinsci*. Additional download locations listed in *Download locations*.

Updating the Synopsys Detect for Jenkins plugin

You can update the Synopsys Detect for Jenkins plugin when new versions are released.

To update the Jenkins plugin:

1. Navigate to **Manage Jenkins** > **Manage Plugins**.
2. Click the **Updates** tab.
3. Select **Synopsys Detect**
   a. If there are updates for the Synopsys Detect for Jenkins plugin, the updates display in the list. If there is not an available update, the Synopsys Detect for Jenkins plugin does not display in this list.
   b. Alternatively, you can force Jenkins to check for plugin updates by clicking **Check now** on the **Updates** tab.
4. If there are updates, select the one you want, and click **Download now and install after restart**.

Configuring the Jenkins Plugin

Use the following process to configure the Synopsys Detect for Jenkins plugin. Note that the supported credential formats are user name and password or API token. SAML is not supported.

1. After installing, navigate to **Manage Jenkins** > **Configure System**.
2. Navigate to the **Synopsys Detect** section, and complete the following.
   a. **Global download strategy**: Depending on your desired deployment method, select either the option to **Install Air Gapped Detect as a Tool Installation** or **Download via scripts or use DETECT_JAR** from the drop-down list.

Figure 1. Global download strategy Air Gap.

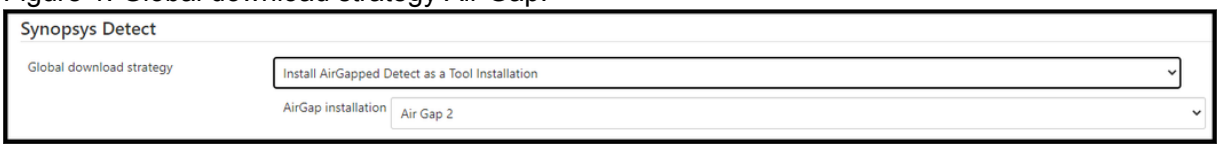
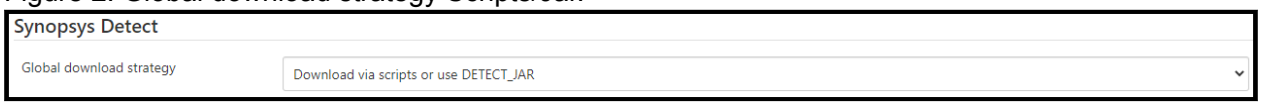


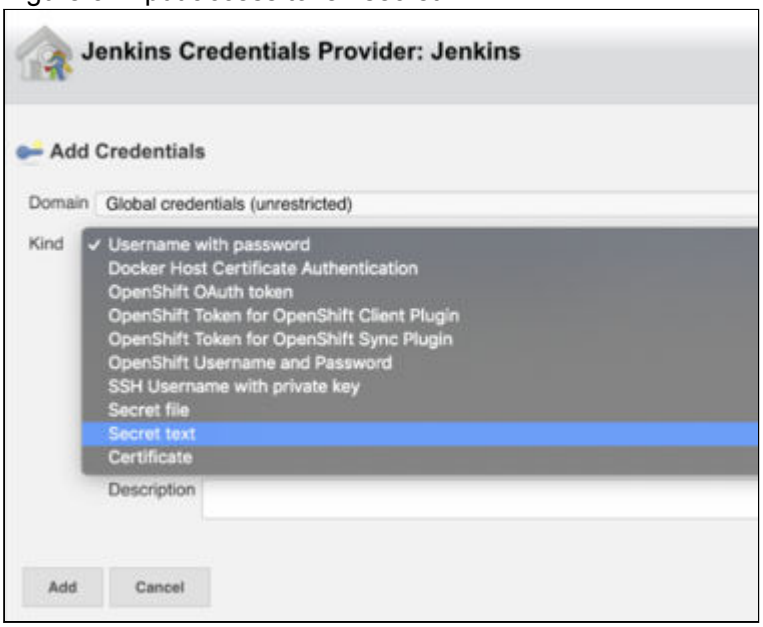
Figure 2. Global download strategy Scripts/Jar.



1. **Black Duck URL**: URL to your Black Duck server instance.
2. **Black Duck credentials**: To add credentials, click **Add** > **Jenkins**, and then select the type of credentials that you want to add and populate the relevant fields. When you add credentials, you can select those credentials that you want from the drop-down menu to authenticate to the Black Duck server.

   a. For user API tokens, select **Secret text** from the menu in the **Kind** field, then provide your Black Duck access token in the **Secret** field.

   Figure 3. Input access token secret.

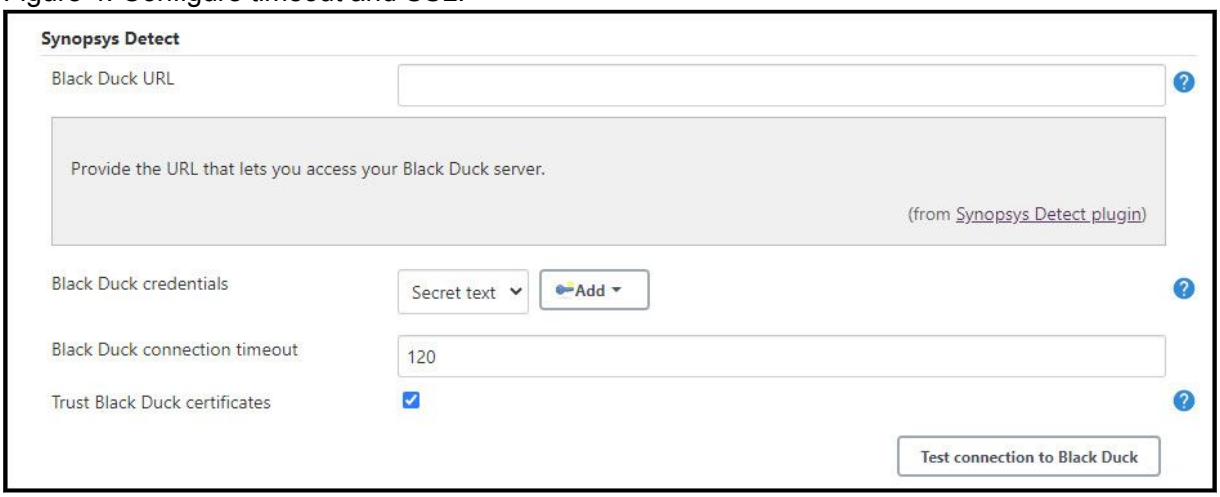


1. The **Advanced...** option displays for Black Duck. Advanced settings enable you to specify values for:

   a. **Black Duck connection timeout** (in seconds). The default value is 120.

   b. **Trust Black Duck certificates**: Select the checkbox to allow (SSL) certificates from Black Duck.

Figure 4. Configure timeout and SSL.



Users and Roles for Jenkins Plugin

First you must configure a user/API token in Black Duck so that the Synopsys Detect is analyzed in Black Duck.

Generating an API token

1. Log in into your Black Duck instance.
2. From the user menu located on the top navigation bar, select My Access Tokens. The **My Access Tokens** page appears.
3. Click **Create New Token**. The Create New Token dialog box appears
4. Type your name in the **Name** field.
5. Optional: in the **Description** field, you can type a description or definition.
6. Select **Read Access** and/or **Write Access**.
7. Click **Create**. The API token displays in a pop-up window. For security reasons, this is the only time your user API token displays. Please save this token. If the token is lost, you must regenerate it.
8. Optional: To modify an access token that you created, click the arrow in the same row as the access token name to open a drop-down menu and select **Edit**, **Delete**, or **Regenerate**.
9. Configure the plugin with your Black Duck url and the API token you just generated.

The following user roles are required for the user that you create in Black Duck:

Role	Action
Project Creator	Creates Black Duck projects

Role	Action
Project Code Scanner	Populates project BOMGlobal Code Scanner can also be used to populate Project BOM
Global Code Scanner	Populates the project BOM, generates reports, checks for policy violations.
Project Manager	Generates reports

Note: A user with the Global Code Scanner overall role can generate a report, but cannot delete the report. The Project Manager project role is required to delete the report.

Running Synopsys Detect in Jenkins

By default, Synopsys Detect for Jenkins downloads either the latest Synopsys Detect shell script when run on a UNIX node, or PowerShell script when it's run on a Windows node, to the Jenkins tools directory, and then executes that script. Note that you can also use the JAR option to run Synopsys Detect.

The Synopsys Detect PowerShell or shell script is downloaded once and placed in the Synopsys Detect working directory. If you want to force the plugin to fetch the latest script, clear out the Detect directory in your Jenkins tools directory.

JAR option

If you do not want to download Synopsys Detect, you can manually put the JAR on the node where you want Synopsys Detect to run and specify the DETECT_JAR environment variable that points to your provided JAR, and that JAR will be executed instead.

To use the JAR option, perform the following steps:

1. Navigate to **Dashboard > Manage Jenkins > Configure System > Global properties > Environment variables**.
2. Click **Add**.
3. Set an environment variable with the following properties:
   a. **Name**: `DETECT_JAR`.
   b. **Value:** `<path to the Detect jar file on your Jenkins node>`.

Note: When your build runs, Jenkins looks for configured environment variables, and if it locates DETECT_JAR, it uses that instead of pulling the latest Synopsys Detect shell script.

Air Gap option

Synopsys Detect can be configured to run in an air gap fashion, see: *Air Gap*.

In freestyle and Pipeline jobs, you can toggle between the different modes for running Synopsys Detect in the plugin such as pulling the Detect.jar from scripts or $DETECT_JAR_PATH, or from a specified Tool Installation.

Running Detect in a job

You can run Synopsys Detect as a post-build action or a Pipeline step.

Pipeline step

You can configure the scan as a pipeline step in a Pipeline job.

Refer to the *pipeline example*

Post-build actions

You can configure the scan as a post-build action in a freestyle job. You can have multiple post-build actions, but only one Synopsys Detect post-build action.

Refer to the *freestyle example*.

DSL considerations

The Synopsys Detect for Jenkins plugin provides Dynamic DSL for both freestyle steps and pipeline steps. Read more at *Dynamic DSL*.

Note: that versions 1.72 and later of the DSL plugin do not support the Synopsys Detect for Jenkins plugin pipeline steps.

Detect in Jenkins Pipeline job

In pipeline jobs there are only steps. You can generate the Synopsys Detect pipeline step as follows.

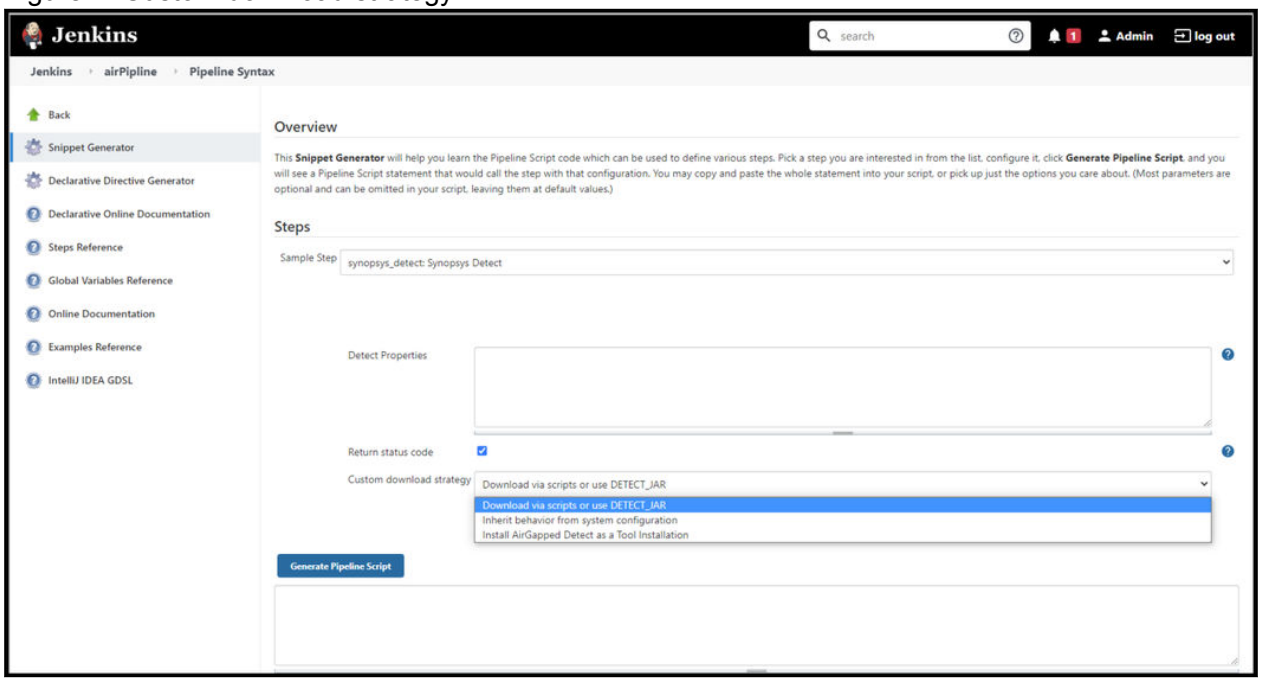
1. Navigate to **Jenkins > New Item**.
2. In the **Enter an item name** field, type the name for your new Pipeline project.
3. Scroll down and click **Pipeline**.
4. Click **OK**.
5. On the resulting page, click the **Pipeline** tab.
6. To help you generate Pipeline syntax, in the **Pipeline** section, click **Pipeline Syntax** to access the **Pipeline Syntax** page.
   a. On the **Pipeline Syntax** page, click the **Sample Step** drop-down menu under **Steps**, and select **synopsys_detect: Synopsys Detect**
      i. Add some Detect properties.

ii. Click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about.

iii. **Optionally**, select the **Return status code** checkbox to return a status code.

iv. **Optionally**, select a **Custom download strategy** option.

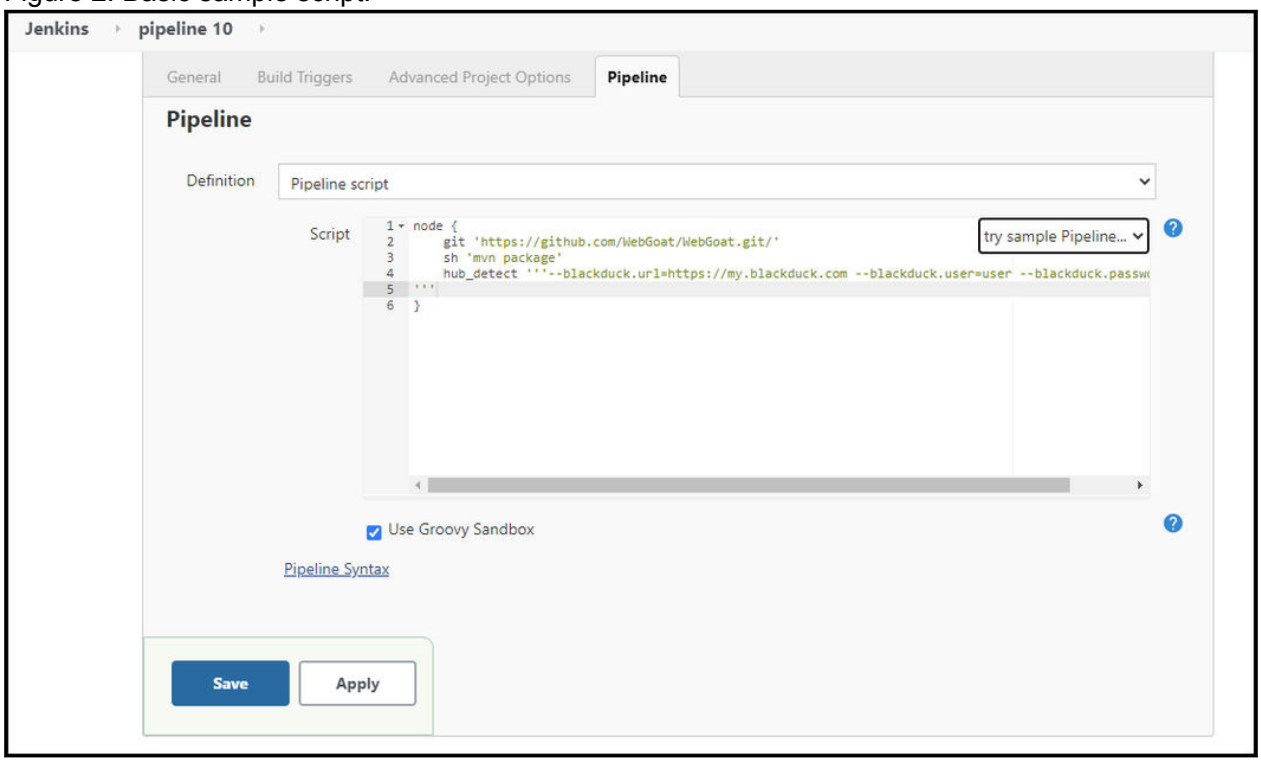
Figure 1. Custom download strategy.



1. Add a Pipeline script and click **Save**.

The following is a simple example of a basic script.

Figure 2. Basic sample script.



1. Run the build.
2. After completing the Jenkins Pipeline build with Synopsys Detect, you can view the complete scan results in your Black Duck instance.

Note: In Jenkins pipelines, there are no post-build actions because post-build actions are a Freestyle job concept.

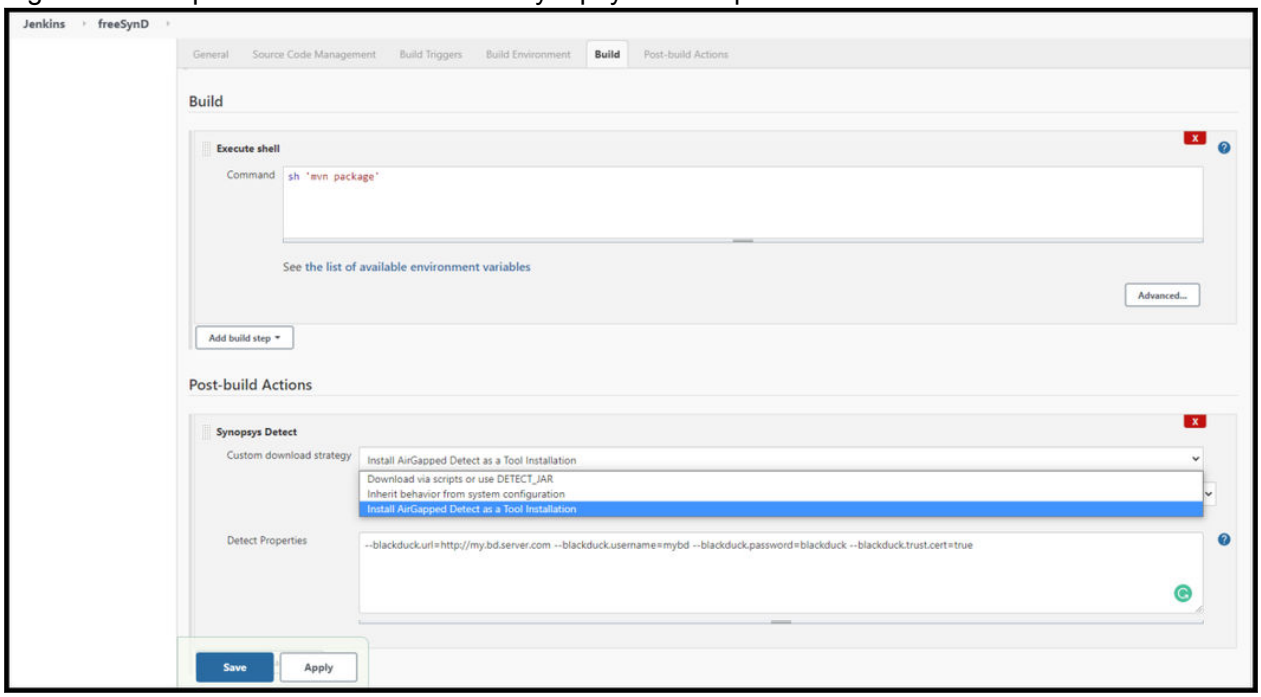
Detect in Jenkins Freestyle job

In a FreeStyle job, you can use post-build actions.

1. Configure your FreeStyle job.
2. Configure your build.
3. Configure a post-build Action.
4. Optionally, select a **Custom download strategy** to run Synopsys Detect.

   The following example shows a simple build command with a Synopsys Detect post-build action.

Figure 1. A simple build command with a Synopsys Detect post-build action.



Auto-escaping Parameters

In Synopsys Detect for Jenkins, several special parameters are automatically escaped. The workflows pertaining to quotation marks and spaces are as follows.

- Detect properties must be separated by spaces or carriage returns/line feeds.
- Values containing spaces must be surrounded by either single or double quotation marks ('single' or “double”).
- Values containing single quotes must be surrounded with double quotation marks.
- Values containing double quotes must be surrounded with single quotation marks.

Considerations for name escaping conventions

You can turn off auto escaping by setting the environment variable *DETECT_PLUGIN_ESCAPING* to false. Jenkins enables you to set an environment variable at different levels, such as globally or on a per-job basis. If you set the environment variable globally to one value, you can set it at the job level to another value. Synopsys recommends setting the environment variable globally to skip escaping (ensuring past jobs work as expected), and then if you want to make jobs with the auto escaping enabled, you modify the environment variable flag in that job's configuration to enable escaping the characters. The easiest way to accomplish this is to install the *"Environment Injector" Jenkins plugin*.

Note: In Synopsys Detect plugin version 8, the above recommendations remain the same for agents on Windows systems. For those agents running on 'NIX systems, *DETECT_PLUGIN_ESCAPING* should be set to false. Ensure that you adhere to the quoting conventions described above. Any input with spaces in the Jenkins configuration should be enclosed in quotes.

For users of Synopsys Detect for Jenkins versions 2.0.0/2.0.1 that are upgrading: if you escaped your values, your escape characters are escaped and appear literally.

For example, you have a job that was created in versions 2.0.0 or 2.0.1. In Synopsys Detect for Jenkins version 8.0.0, spaces are not allowed; therefore they must be escaped to have a name with spaces. If the project name is My Test Project1, you must pass the project name as `My\ Test\ Project1`. Hence, the project is uploaded in Black Duck as `My Test Project1*.*`

Synopsys Detect for Jenkins allows some special characters, and spaces can be included without escape sequences. Therefore, instead of `My\ Test\ Project1`, you can pass it as My Test Project1 and the project is created successfully and uploaded to Black Duck as `My Test Project1*.*`

If there are jobs that are already configured in 2.0.1 as `My\ Test\ Project2` and after upgrading the Synopsys Detect for Jenkins, these existing projects are uploaded to Black Duck as `My\ Test\ Project2*.*`

Note: that this change has an impact on jobs created in version 2.0.1 with existing project names containing escape sequences and then upgrading Synopsys Detect for Jenkins to version 2.0.2+.

If there are existing jobs created with escape sequences, the impact of this change can be greater.

Jenkins Air Gap mode

The Synopsys Detect for Jenkins plugin enables you to configure an Air Gap option to run Synopsys Detect.

Before you can see the Detect Air Gap option on the Global Tool Configuration page, you must install the Synopsys Detect plugin.

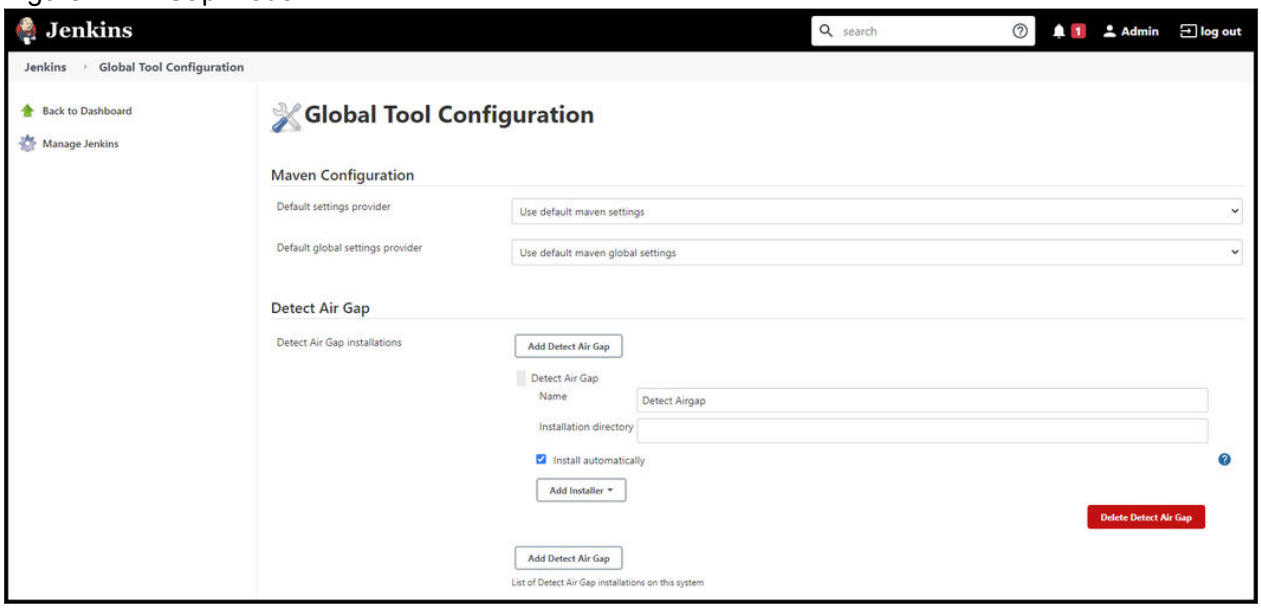
Use the following process to make the Air Gap option globally available when you're configuring a Synopsys Detect job:

1. In Jenkins, Click **Manage Jenkins** on the left navigation and then click **Global Tool Configuration**.
2. In the Detect Air Gap section, click **Add Detect Air Gap** and then complete the following:
   a. **Detect Air Gap Name**: A name for the Air Gap installation.
   b. **Installation directory**: The directory for the Air Gap installation files.
   c. **Install automatically**: Select this checkbox to enable Jenkins to install the Air Gap files on demand.

When you check this option, you have to configure an installer for this tool, where each installer defines how Jenkins will attempt to install this tool.

For a platform-dependent tool, multiple installer configurations enable you to run a different setup script depending on the agent environment, but for a platform-independent tool such as Ant, configuring multiple installers for a single tool wouldn't be suggested.

Figure 1. Air Gap mode.



1. Optionally, add another Air Gap version. You can use the **Add Installer** menu to choose other install methods such as **Run Batch Command** or **Run Shell Command**.
2. Click **Save**.

Using Docker Containers - Best Practice

When running Synopsys Detect for Jenkins using the *DETECT_JAR* environment variable in a pipeline that has a Docker agent, remember to mount the Detect JAR as a volume.

For example, if you've installed the jar on a node, the JAR won't be accessible from your docker agent unless you either put it somewhere you regularly include, or if you mount the path to the jar. Refer to *Docker documentation* for more information.

This is accomplished by adding: `-v $DETECT_JAR:$DETECT_JAR` to your Docker arguments, which is shown in the following example.

```
pipeline {
    agent {
        docker {
            ...
            args '-v $DETECT_JAR:$DETECT_JAR'
        }
```

```
    . . .
. . .
```

Applies when running Synopsys Detect for Jenkins with a local JAR (which requires setting the *DETECT_JAR* environment variable).

Azure DevOps (ADO) Plugin

The Synopsys Detect for Azure DevOps plugin, formerly known as Black Duck Detect plugin for TFS/VSTS, is architected to seamlessly integrate Synopsys Detect with Azure DevOps build and release pipelines. Synopsys Detect makes it easier to set up and scan code bases using a variety of languages and package managers.

The Synopsys Detect plugin for Azure DevOps supports native scanning in your Azure DevOps environment to run Software Composition Analysis (SCA) on your code.

As a Synopsys and Azure DevOps user, Synopsys Detect Extension for Azure DevOps enables you to:

- Run a component scan in an Azure DevOps job and create projects and releases in Black Duck through the Azure DevOps job.
- After a scan is complete, the results are available on the Black Duck server (for SCA).

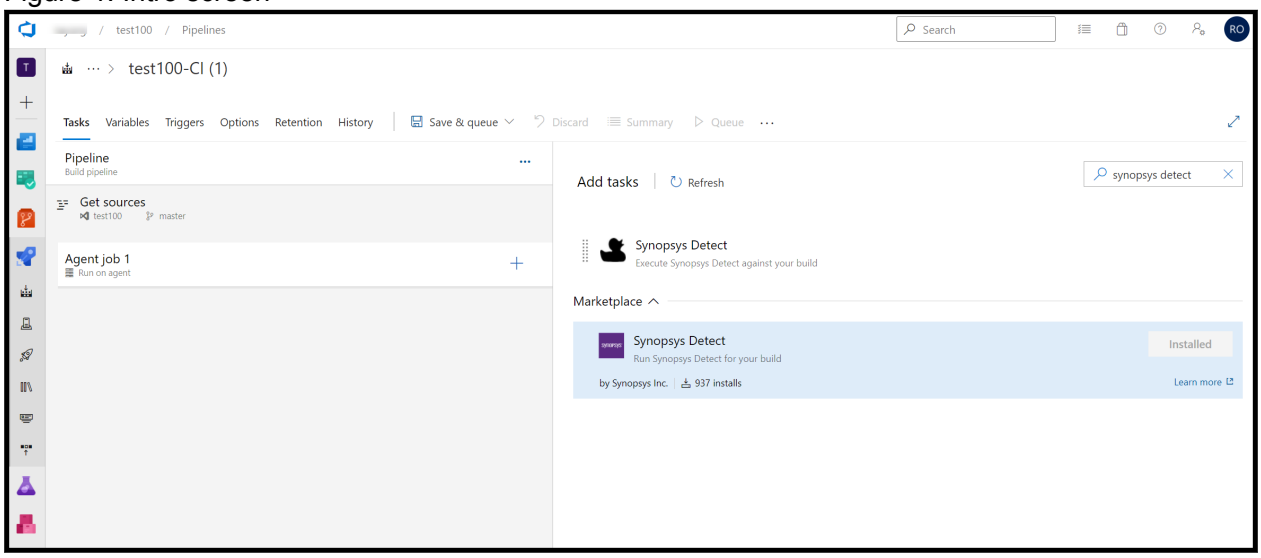
Using the Synopsys Detect Extension for Azure DevOps together with Black Duck enables you to use Azure DevOps to automatically create Black Duck projects from your Azure DevOps projects.

Note: The Azure plugin currently supports Synopsys Detect \8.

Invoking Synopsys Detect

Synopsys recommends invoking Synopsys Detect from the CI (build) pipeline. Scanning during CI enables Synopsys Detect to break your application build, which is effective for enforcing policies like preventing the use of disallowed or vulnerable components.

Figure 1. Intro screen



Basic workflow

Using Synopsys Detect to analyze your code in Azure involves the following basic steps:

1. Make sure you satisfy system and other requirements.
2. Download and configure the Synopsys Detect extension in Azure.
3. Configure build agent and pipeline.
4. Configure Black Duck connection.
5. Configure Synopsys Detect arguments.
6. Run pipeline and invoke scan.
7. Examine the analysis results.

Release Notes for Azure DevOps Plugin

Version 8.1.0

New features

- (DETECTADO-95) Plugin is now able to inherit the Azure agent's proxy configuration.
  - Refer to *Configuring a Build Agent with a proxy* for more information.

Version 8.0.0

New features

- Updated the plugin to use Synopsys Detect 8.

Version 7.0.0

New features

- Updated the plugin to use Synopsys Detect7.
- Added the ability to run Synopsys Detect in AirGap mode.

Version 6.0.0

Resolved issues

- (DETECTADO-68) Improved error messaging when invalid proxy details are used.
- (DETECTADO-70) Resolved issue wherein passing properties on new lines would cause Detect ADO to fail.
- (DETECTADO-71) Resolved issue with TLS errors being thrown on Windows hosted agents.

Changed features

- The plugin versioning was changed to match the major version of Synopsys Detect that it is designed to work with, for example Detect ADO 6.0.0 works with Synopsys Detect major version 6.

Version 3.0.0

New features

- Added the capability for the script to use the tool directory in the ADO agent to store the Synopsys Detect JAR. It will continue to use this JAR as long as the JAR version matches the version specified in the task configuration.
- Added support for using Linux and Mac agents.

Changed features

- Removed support for Polaris.

Version 2.0.0

New features

- Added support for Polaris.

Changed features

- Product renamed to Synopsys Detect for Azure DevOps.

Version 1.1.0

Changed features

- The service endpoint configuration is now optional.
- Added support for using an API token for user authentication.

Version 1.0.4

Changed features

- Improved proxy support and handling of supplied proxy arguments.

Resolved issues

- Resolved an issue that could result in an *Access denied* error.

Version 1.0.3

Resolved issues

- Resolved an issue involving the SSL issue casting protocol.

Version 1.0.0

- First release of product

Requirements for Azure DevOps

The following is a list of requirements for the Synopsys Detect in Azure DevOps integration.

- Black Duck server. For the supported versions of Black Duck, refer to *Black Duck Release Compatibility*.
- Black Duck API token to use with Azure.
- Azure DevOps Services or Azure DevOps Server 17 or later
- Java. OpenJDK versions 8 and 11 are supported. Other Java development kits may be compatible, but only OpenJDK is officially supported for Black Duck.
- Access to the internet is required to download components from GitHub and other locations.

The Synopsys Detect plugin for Azure DevOps is supported on the same operating systems and browsers as Black Duck.

For scanning NuGet projects, verify that you have the NuGet tool installer set up in the build job definition. For further information see *NuGet tool*

You can get the Synopsys Detect for Azure DevOps plugin at *VisualStudio Marketplace*.

Installing the Azure DevOps plugin

From the Azure Pipelines page, add the Synopsys Detect plug-in for ADO.

Install the Synopsys Detect extension for Azure DevOps

1. Click the plus sign (**+**) under Tasks for Agent Job
2. Search for the Synopsys Detect plugin and click **Add** to add it to your pipeline.

Figure 1. Adding the plugin

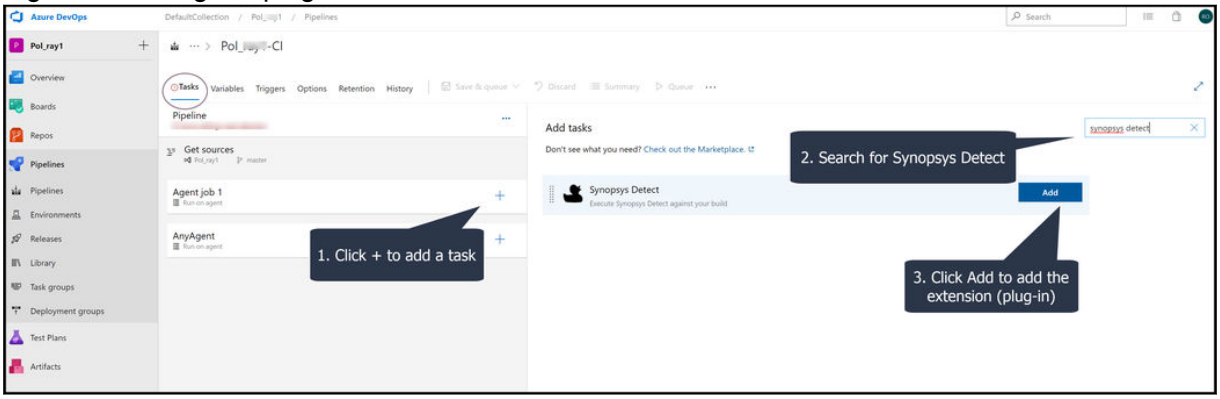
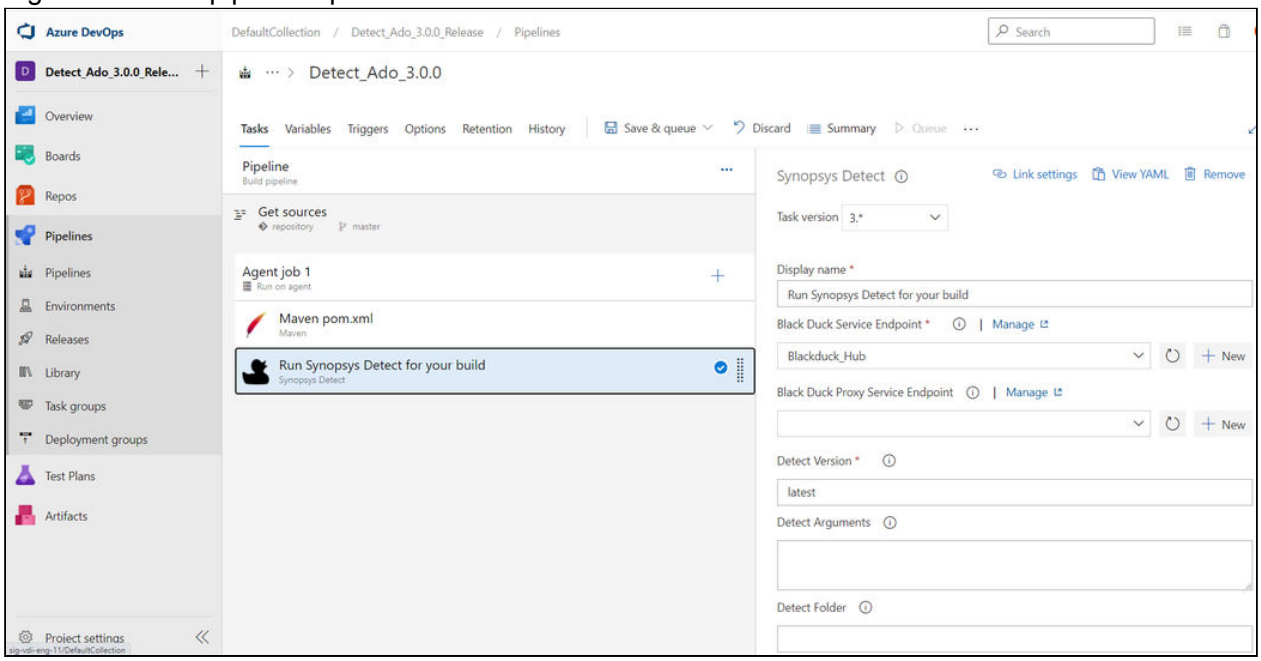


Figure 2. Now a pipeline option

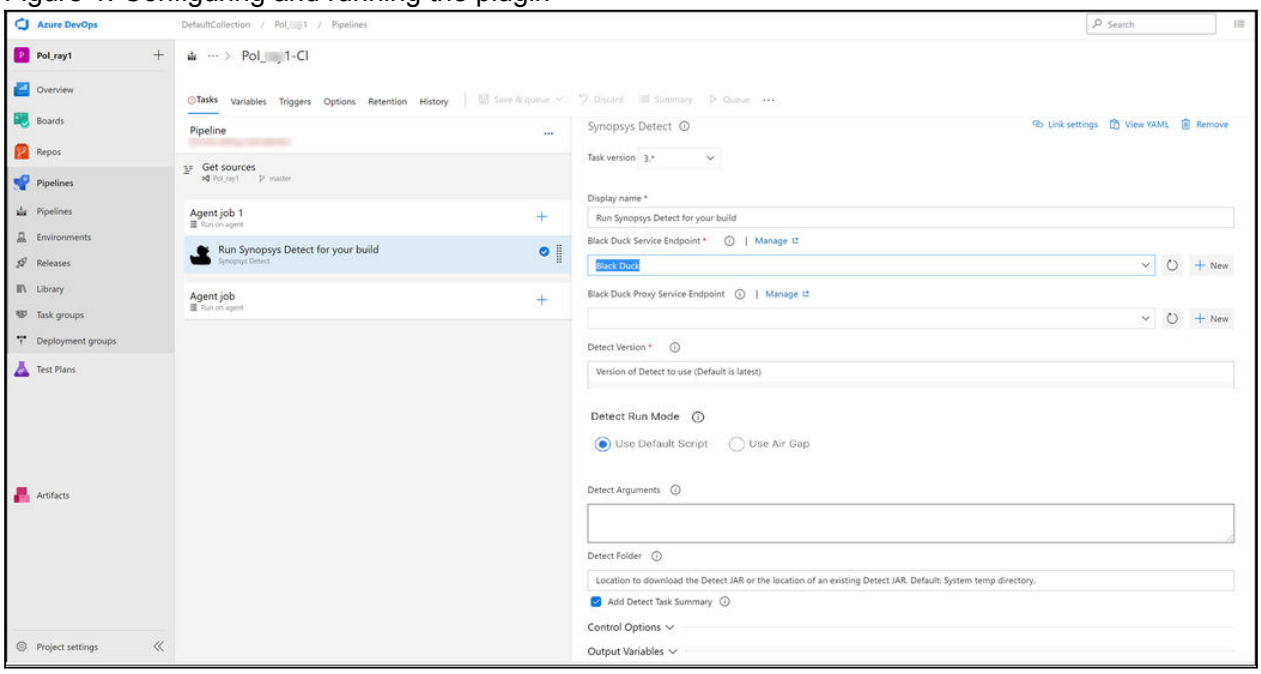


Configuring and Running the Plugin

After you install the plugin, you configure it in Pipeline task.

Configure your Synopsys Detect for Azure DevOps plugin by adding configuration for your Black Duck server and adding Detect arguments.

Figure 1. Configuring and running the plugin



Configuring the plugin

1. Navigate to **Your Collection > Project > Pipelines > Tasks**. The plugin adds a new task of **Run Synopsys Detect for your build**. You must add this task to your build queue.
2. Click **Run Synopsys Detect for your build**, and the **Synopsys Detect** panel displays on the right. In the **Synopsys Detect** configuration panel, complete the following fields and options.
3. **Display name:** Type a unique name in this field. Note that the name you type here displays in the left panel; the default name is **Run Synopsys Detect for your build**.
4. Click **+ New** to add a new **Black Duck Service Endpoint** and then configure the details.
5. Click **+ New** to add a new **Black Duck Proxy Service Endpoint** and then configure the details.
6. **Detect Version**: Version of the Synopsys Detect binary to use. Synopsys recommends using the latest but you can specify a version override if desired.
7. **Detect Run Mode:** Select the run mode. If you select Use Airgap Mode, a Detect Air Gap Jar Directory Path field opens in which you must specify the Synopsys Detect Air Gap Jar Path.
8. **Detect Arguments**: Here you can include additional Synopsys Detect* arguments; Synopsys Detect picks up your build environment variables and your project variables. Use a new line or space to separate multiple arguments. Use double quotes to escape. You can use environment and build variables. For more information on Synopsys Detect arguments, refer to *Properties*.

9. **Detect Folder**: The location to download the Detect jar or the location of an existing Detect jar. The default is the system temp directory. To specify a different directory, type the directory path and name in the field.

Windows agents require an absolute path when specifying detect download location in the **Detect Folder** field.

1. **Add Detect Task Summary**: Click this checkbox to add a summary of the Detect task to the build summary task.

In the user interface, fields with a red asterisk (*) are required. Some default values are provided, such as version.

Note: that the following fields belong to Azure DevOps, and are not part of the Detect plugin:

- Task version
- Display name
- Control Options
- Output Variables

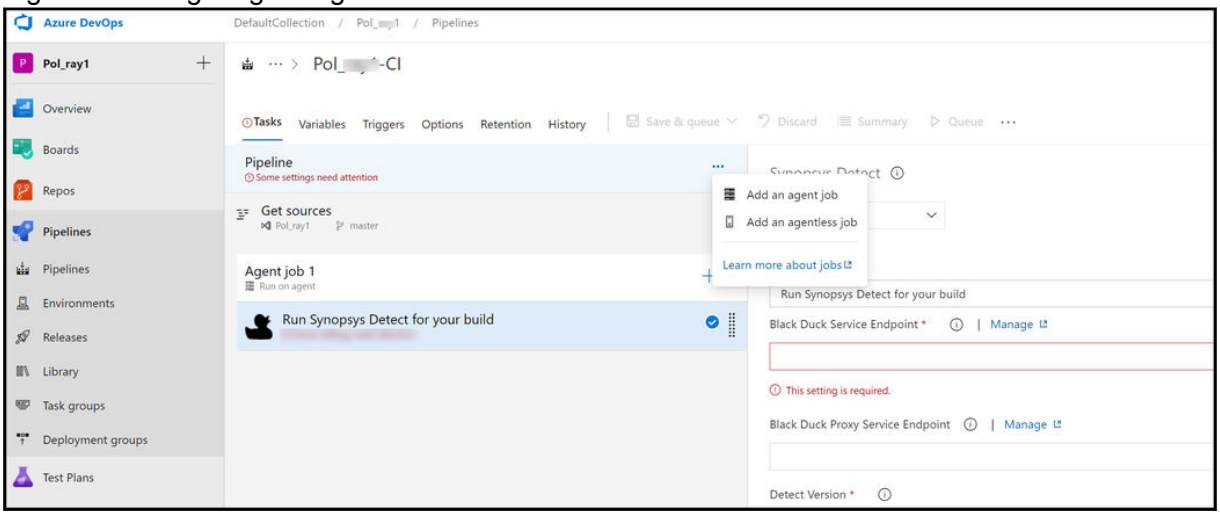
Configuring a Build Agent

To configure a build agent in your pipeline do the following under the **Tasks** tab on your pipeline page.

The default option for the build agent is the Microsoft hosted agent. To be able to select a self-hosted agent, you must have installed the agent and ensure that it's available to your project before you can use it in your pipeline. Click the ellipsis (**...**) next to Pipeline to Add an agent job.

1. Click the ellipsis (**...**) next to Pipeline to Add an agent job.

Figure 1. Configuring an Agent



1. On the Agent job configuration screen, do the following:

a. Select a self-hosted agent from your Agent pool or select Azure Pipelines for an Azure-hosted agent.
b. If you select a hosted agent, then you must select an operating system such as macOS, Windows, or a version of Linux for the hosted agent VM.

Tip: This is not an airgap option as internet connections are still required for downloading other tools and the script will still download new content if needed.

Note: If the agent is behind a proxy, Synopsys Detect Azure plug-in will utilize the agent proxy by default.

Configuring with a proxy

You can configure the build agent for Synopsys Detect Azure Plugin to use a proxy when running jobs.

Proxy configuration scenarios

1. If both an agent proxy and Black Duck Poxy Service Endpoint are set through ADO Plugin parameter, the Black Duck proxy url endpoint takes precedence.
2. If agent proxy is configured, and the Black Duck Poxy Service Endpoint is not set through ADO Plugin parameter, the Synopsys Detect Azure Plugin utilizes the agent proxy.

Running the Task

After you have configured your task, you can run it as follows:

- In Azure DevOps, click **Queue**, and your task is executed on the next available build agent.
- If your task configuration is incomplete, a red status message of "*Some settings need your attention*" displays below **Run Synopsys Detect for your build**. Missing required settings display in red in the **Synopsys Detect** panel.

GitLab Integration

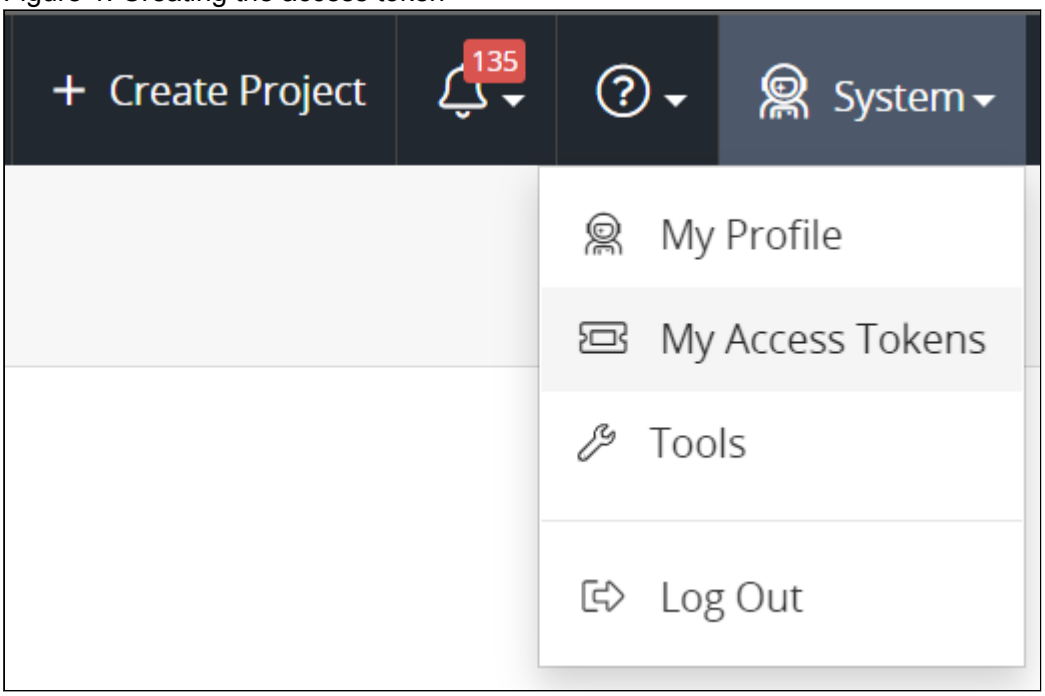
Synopsys Detect is designed to run in the native build environment of the project you want to scan. The following procedures provide guidance on setting up Synopsys Detect with your GitLab continuous integration builds.

Configuring with API tokens

The recommended way of configuring Synopsys Detect from a GitLab pipeline is to use an API token. This is detailed as follows.

1. In Black Duck, navigate to the profile of the user whose credentials are used to scan projects from the pipeline.
2. Scroll down to the **User Access Token** section, and complete the fields to create a new token.
3. Check both the **Read Access** and **Write Access** boxes.
4. Click **Generate.** Save or copy the displayed token.

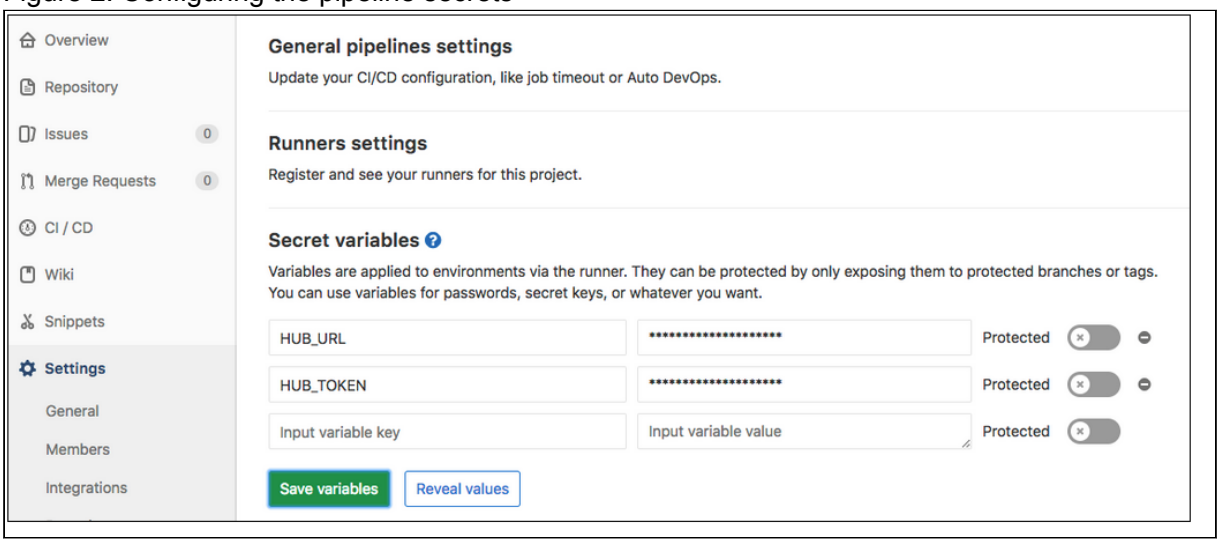
Figure 1. Creating the access token



Configuring your environment variables

1. In the sidebar, navigate to **Settings**. Then select **CI/CD**.
2. Expand the **Secret variables** tab.

Figure 2. Configuring the pipeline secrets



3. Create two environment variables:
   - HUB_URL - containing the URL of your Black Duck installation.
   - HUB_TOKEN - containing the API token generated in the prerequisite steps.

   Note: You can make these variables protected. For additional information, refer to *Gitlab protected secret variables*.
4. Configure Synopsys Detect to be a script step in the *.gitlab-ci.yml* file of the project you want to scan. Then add the snippet for Synopsys Detect.

   Note: Ensure that the final line of the following command fits on a single command line.

```
        image: java:8build:
                stage: build
                script:
                - ./gradlew assemble
        test:
                stage: test
                script:
                - bash <(curl -s -L https://detect.synopsys.com/detec
t8.sh) --blackduck.url="${HUB\_URL}" --blackduck.api.token={your Blac
k Duck access token} --blackduck.trust.cert=true --<any other flags>
```

5. Configure Synopsys Detect as a script build step. Otherwise, GitLab cannot enforce build changes influenced by Synopsys Detect. For example, checking for policy, failing builds according to policy, and others.

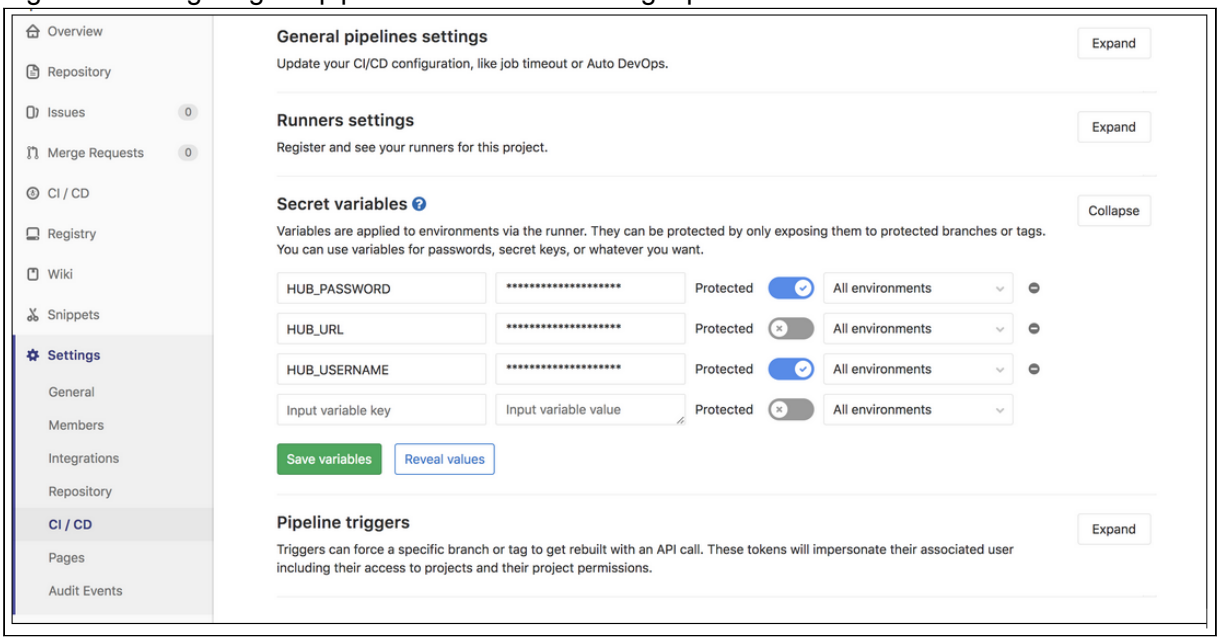
6. After you commit the change to *.gitlab-ci.yml,* the pipeline runs. After the build with Synopsys Detect completes, you can view the complete scan results in your Black Duck instance.

Configuring with username and password

For improved security, Synopsys recommends a revocable API token, as described in the preceding process, instead of storing an account password in GitLab settings.

1. In the sidebar project menu, navigate to **Settings** Then select **CI/CD**.
2. Expand the **Secret variables** tab.

Figure 3. Configuring the pipeline secret variables/figcaption>



3. Create three environment variables:
   - HUB_URL - containing the URL of your Black Duck environment.
   - HUB_USERNAME - containing the username of the Black Duck account to be used.
   - HUB_PASSWORD - containing the password of the Black Duck account to be used.

   Note: You can make these variables protected. For additional information, refer to *Gitlab protected secret variables*.
4. Configure Synopsys Detect to be a script step in the *.gitlab-ci.yml* file of the project you want to scan. Then add the snippet for Synopsys Detect.

   Note: Ensure that the final line of the following command fits on a single command line.

```
        image: java:8build:
                stage: build
                script:
                - ./gradlew assemble
        test:
                stage: test
                script:
                        - bash <(curl -s -L <https://detect.synopsys.
com/detect8.sh>) --blackduck.url="${HUB\_URL}"  --blackduck.hub.usern
ame="${HUB\_USERNAME}" --blackduck.hub.password="${HUB\_PASSWORD}" --
blackduck.api.token={your Black Duck access token} --blackduck.trust.
cert=true --<any other flags>
```

5. Configure Synopsys Detect as a script build step. Otherwise, GitLab cannot enforce build changes influenced by Synopsys Detect. For example, checking for policy, failing builds according to policy, and others.
6. After you commit the change to *gitlab-ci.yml*, the pipeline runs. After the build with Synopsys Detect completes, you can view the complete scan results in your Black Duck instance.

July 13, 2023