Synopsys Bridge CLI Guide

Contents

Chapter 1. Overview

Overview

Use any of these to integrate Static Analysis Security Testing (SAST) and Software Composition Analysis (SCA) into your CI/CD pipelines:

- Synopsys Bridge CLI
- Synopsys Action for GitHub
- Synopsys Template for GitLab
- Synopsys Security Scan for Azure DevOps

With any of the integrations above you can:

- Scan when you merge code or on a pull request.
- Optionally create pull request comments when new issues are found.
- Optionally create a new pull request to automatically update vulnerable components.

Further information

- Support Matrix *(on page 4)*
- Files and Directories *(on page 5)*
- Download Synopsys Bridge *(on page 5)*

Support Matrix

The table below outlines which Synopsys security tools are supported by Synopsys Bridge.

Tool	Bridge Support?	Notes
Polaris	Yes	Polaris users can use Synopsys Bridge CLI to automate SAST and/or SCA scans in their CI pipeline. Click here for SAST specific system requirements.
Black Duck	Yes	Black Duck users can use Synopsys Bridge CLI to automate SCA scans in their CI pipeline.

Tool	Bridge Support?	Notes
Coverity Connect	Yes	Coverity users can use Synopsys Bridge CLI to automate SAST scans in their CI pipeline. Synopsys Bridge can be used with both an on-prem Coverity Connect as well as Coverity cloud deployment. Click here for system requirements.

Synopsys Bridge runs on the following operating systems:

OS	System Requirements	Notes
Linux	64-bit kernel, version 2.6.32+ with glibc 2.18 or later	Debian GNU is *not* supported
macOS	OSX 11, 12, 13	macOS 11, 12 and 13 on Intel (M1 and M2 based Macs are not currently supported)
Windows	x86_64, Version 10 and 11 and Windows Server 2019 and 2022	Server Core is *not* supported

Files and Directories

By default, the Synopsys Bridge Bridge writes logs and temporary files to `<current_working_directory>/.bridge`. You may change this default directory by using the `--home <directory_path>` option.

The following files and directories are found under the Synopsys Bridge home directory:

- `bridge.log`
- `diagnostics.json` file with `--diagnostics` option. See Logging and Diagnostics *(on page 33)* for details.
- Adapter directories and the corresponding `stdout` and `stderr` log files.
- Additional temporary files.

Download Synopsys Bridge

You can download the latest version of Synopsys Bridge from Synopsys Artifactory.

Polaris users can also download Synopsys Bridge directly from the Polaris user interface:

- Click **username** at the top right.
- Select **Accounts**.
- Select **Downloads**.
- Choose the appropriate package for your operating system.

To install, simply unzip and add `synopsys-bridge` executable to your PATH or use absolute PATH to `synopsys-bridge` executable.

Chapter 2. Synopsys Bridge CLI

Using Synopsys Bridge CLI

Once you have `synopsys-bridge` executable installed, you are ready to use Synopsys Bridge to integrate SAST and/or SCA scans into your CI/CD pipeline.

You can run Synopsys Bridge in one of the following two ways:

1. By passing arguments through a JSON file.
2. By passing arguments on the command line.

For a complete list of exit codes returned by Synopsys Bridge, see the Exit Code *(on page 33)* table.

Passing Arguments using a JSON file

Passing arguments using a JSON file greatly simplifies the command line and promotes reuse. Here are the steps:

1. Create an access token in the web interface of the Synopsys security product you are integrating with.
2. Use environment variable(s) to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Synopsys Bridge automatically picks up values passed through these variables.
    - Example: `export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>`.
3. Pass the JSON file to Synopsys Bridge using the `--input` command line option.
4. Pass the Synopsys security product you are integrating using the `--stage` option.

Here are the example commands::

```
export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>

synopsys-bridge --stage polaris --input input.json
```

Note:
Depending on your OS, you will need to use appropriate mechanism to set environment variables.

Here is the `input.json` file:

```
{
 "data": {
```

```
  "polaris": {
   "application": {
    "name": "<APPLICATION_NAME>"
   },
   "project": {
    "name": "<PROJECT_NAME>"
   },
   "assessment": {
    "types":  ["SAST", "SCA"]

   },
   "serverUrl": "<SERVER_URL>"
  }
 }
}
```

Note:
It is recommended that you save the JSON file at the root of the project directory being scanned. The JSON file can have any name as long as it has a `.json` extension.

Note:
You can use different JSON files for different use cases.

Note:
For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

For tool specific information and examples, see:

- Using Synopsys Bridge with Polaris *(on page 9)*
- Using Synopsys Bridge with Black Duck *(on page 12)*
- Using Synopsys Bridge with Coverity Connect *(on page 15)*

Passing Arguments using the CLI

You can also pass arguments on the command line as an alternative to passing arguments using a JSON file.

Here are the steps:

1. Create an access token in the web interface of the Synopsys security product you are integrating with.
2. Use environment variable(s) to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Synopsys Bridge automatically picks up values passed thru these variables.
    - Example: `export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>`
3. Pass the necessary command line arguments as shown in the example below.

```
export BRIDGE_POLARIS_ACCESSTOKEN="<POLARIS_ACCESSTOKEN>"

synopsys-bridge --stage polaris polaris.project.name="<PROJECT_NAME>" \

polaris.application.name="<APPLICATION_NAME>" \

polaris.assessment.types=SAST,SCA \

polaris.serverUrl="<POLARIS_SERVERURL>"
```

For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

See Schema Resources And Extensions *(on page)* for Synopsys Bridge resources.

For tool specific information and examples, see:

- Using Synopsys Bridge with Polaris *(on page 9)*
- Using Synopsys Bridge with Black Duck *(on page 12)*
- Using Synopsys Bridge with Coverity Connect *(on page 15)*

Using Synopsys Bridge CLI with Polaris

As a Polaris customer, you can use Synopsys Bridge to automate SAST and SCA scanning in your CI/CD pipeline.

You can use Synopsys Bridge to run Polaris scans in the following two ways:

- Running Polaris scans with a JSON file *(on page 10)*
- Running Polaris scans on the command line *(on page 11)*

In addition to running scans, you can also optionally configure Synopsys Bridge to create fix pull requests for SCA issues. Currently, only NPM is supported. For more information, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

> **Note:**
> As an alternative to Synopsys Bridge, you can also use Synopsys Github Action *(on page 35)*, Synopsys Template for GitLab *(on page 45)* or Synopsys Security Scan for Azure DevOps *(on page 55)*.

Running Polaris scans with a JSON file

Synopsys Bridge for Polaris uses Coverity for SAST scans and Black Duck for SCA scans under the hood. Depending on the task, you may need to pass additional SAST and SCA configurations.

After passing sensitive access token and password information using the `BRIDGE_POLARIS_ACCESSTOKEN` environment variable, run Synopsys Bridge and pass the JSON file using the `--input` command line option. Here is a command line example for Polaris:

```
export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>

synopsys-bridge --stage connect --input input.json
```

The above example uses the following:

- `BRIDGE_POLARIS_ACCESSTOKEN` environment variable to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.

Here is the `input.json` file:

```
{
    "data": {
        "polaris": {
            "application": {
                "name": "<APPLICATION_NAME>"
            },
            "project": {
                "name": "<PROJECT_NAME>"
            },
            "assessment": {
```

```
                "types":  ["SCA", "SAST"]


            },
            "serverUrl": "<POLARIS_URL>"
        }
    }
}
```

The above example uses the following schema resources:

- `polaris.serverUrl` for Polaris URL.
- `polaris.application.name` for Polaris Application to use. Note that the specified application must exist on Polaris with appropriate entitlements.
- `polaris.project.name` for Polaris Project to use. Note that the specified project must exist on Polaris. You can set `polaris.onboarding` to true if you want Synopsys Bridge to automatically create the project in case it doesn't yet exist on Polaris.
- `polaris.assessment.types` specifies the type of scan to be run: SAST or SCA or SAST,SCA.

For the required minimum set of arguments that you need to pass to integrate Synopsys Bridge with Polaris, refer to the Polaris specific resources page under Schema Resources and Extensions *(on page)*.

For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

For additional SAST-specific details, see Additional SAST configuration requirements *(on page 12)*.

Running Polaris scans on the command line

Instead of using a JSON file, you can pass all arguments via the command line. Here is a command line example for Polaris:

```
export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>
synopsys-bridge --stage polaris polaris.project.name="<PROJECT_NAME>" \
polaris.application.name="<APPLICATION_NAME>" \
polaris.assessment.types=SAST,SCA \
polaris.serverUrl="<SERVERURL>"
```

The above example uses the following:

- `BRIDGE_POLARIS_ACCESSTOKEN` environment variable to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.
- `polaris.serverUrl` for Polaris URL.
- `polaris.application.name` for Polaris Application to use. Note that the specified application must exist on Polaris with appropriate entitlements.
- `polaris.assessment.types` specifies the type of scan to be run: SAST or SCA or SAST,SCA.

For the required minimum set of arguments that you need to pass to integrate Synopsys Bridge with Polaris, refer to the Polaris specific resources page underSchema Resources and Extensions *(on page)*.

For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

For additional SAST-specific details, see Additional SAST configuration requirements *(on page 12)*.

Additional SAST configuration requirements

A `coverity.yml` configuration file is required for

- Static analysis of compiled languages like C/C++, C# and Java.
- Optimizing static analysis when results are unsatisfactory.

Certain Coverity Connect scans on Polaris require configuration of additional capture settings using a `coverity.yaml` file. See Configuring Coverity Thin Client for use with Synopsys Bridge and Polaris in the *Polaris Developer Portal* for more information.

Using Synopsys Bridge CLI with Black Duck

As a Black Duck customer, you can use Synopsys Bridge to automate SCA scanning in your CI/CD pipeline.

You can use Synopsys Bridge with Black Duck in the following two ways to run scans:

- Running Black Duck scans with a JSON file *(on page 13)*
- Running Black Duck scans on the command line *(on page 14)*

In addition to running scans, you can also optionally configure Synopsys Bridge to perform the following. For more information, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

- Scan pull requests
- Add comments to pull requests
- Create fix pull requests (NPM only)

Note:
As an alternative to Synopsys Bridge, you can also use Synopsys Github Action *(on page 35)*, Synopsys Template for GitLab *(on page 45)* or Synopsys Security Scan for Azure DevOps *(on page 55)*.

Pass Sensitive Data as Environmental Variables

Before passing arguments with Synopsys Bridge, it is recommended that you pass user name and password arguments using environmental variables for security reasons.

```
export BRIDGE_BLACKDUCK_TOKEN=<BLACKDUCK_TOKEN>
```

Running Black Duck scans with a JSON file

Here is an example command:

```
synopsys-bridge --stage blackduck --input input.json
```

The above example uses the following:

- `BRIDGE_BLACKDUCK_TOKEN` environment variable to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.

Here is the `input.json` file:

```
{
    "data": {
        "blackduck": {
            "url": <BLACKDUCK_URL>,
            "scan": {
                "full": true,
                "failure": {
                    "severities": ["CRITICAL"]
                }
            }
```

```
        }
    }
}
```

The above example uses the following schema resources:

- `blackduck.url` for Black Duck URL.
- `blackduck.scan.full` should be set to true so that Intelligent scans are run by Synopsys Bridge.
- `blackduck.scan.failure.severities` is a list of severities that is used by Synopsys Bridge to decide if the CI pipeline should be failed or not.

For the required minimum set of arguments that you need to pass to integrate Synopsys Bridge with Polaris, refer to the Polaris specific resources page under Schema Resources and Extensions *(on page)*.

For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

Running Black Duck scans on the command line

Instead of using a JSON file, you can pass all arguments on the command line.

Here is a command line example for Black Duck::

```
export BRIDGE_BLACKDUCK_TOKEN=<BLACKDUCK_TOKEN>

synopsys-bridge --stage blackduck \

blackduck.url=<BLACKDUCK_URL> \

blackduck.scan.failure.severities=CRITICAL,HIGH \

blackduck.scan.full=true
```

The above example uses the following:

- `BRIDGE_BLACKDUCK_TOKEN>` environment variable to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.
- `blackduck.url` for Black Duck URL.
- `blackduck.scan.full` should be set to true so that Intelligent scans are run by Synopsys Bridge.
- `blackduck.scan.failure.severities` is a comma separated list of severities that is used by Synopsys Bridge to decide if the CI pipeline should be failed or not.

For the required minimum set of arguments that you need to pass to integrate Synopsys Bridge with Polaris, refer to the Polaris specific resources page underSchema Resources and Extensions *(on page)*.

For a complete list of environment variables and command line arguments, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

Using Synopsys Bridge CLI with Coverity

As a Coverity customer, you can use Synopsys Bridge to automate SAST scanning in your CI/CD pipeline.

> **Note:**
> You can use Synopsys Bridge with both on-prem Coverity Connect as well as Coverity cloud deployment. Details below.

You can integrate Synopsys Bridge with Coverity in the following ways:

- Running Coverity scans using a JSON file *(on page 15)*
- Running Coverity Connect scans using the command line *(on page 17)*

In addition to running scans, you can also optionally configure Synopsys Bridge to add comments to pull requests. For more information, see Complete List of Synopsys Bridge Arguments *(on page 19)*.

> **Note:**
> As an alternative to Synopsys Bridge, you can also use Synopsys Github Action *(on page 35)*, Synopsys Template for GitLab *(on page 45)* or Synopsys Security Scan for Azure DevOps *(on page 55)*.

Running Coverity scans using a JSON file

Before running Synopsys Bridge, it is recommended that you pass user name and password arguments using environmental variables for security reasons.

Here are the example commands:

```
export BRIDGE_COVERITY_CONNECT_USER_NAME="<COV_USER>"

export BRIDGE_COVERITY_CONNECT_USER_PASSWORD="<COVERITY_PASSPHRASE>"

synopsys-bridge --stage connect --input input.json
```

Here's an example `input.json` file that you can use with on-prem Coverity Connect:

```
{
  "data":
    {
        "coverity":
        {
             "connect": {
                    "url": "<Connect URL>",
                        "project":{
                        "name": "<PROJECT_NAME>"
                    },
                    "stream": {
                        "name": "<STREAM_NAME>"
                    },
                    "policy": {
                        "view": "<View Name / Id>"
                    }
                },
            "local": true
        }
    }
}
```

Here is an example `input.json` file that you can use with Coverity cloud deployment:

```
{
  "data":
    {
        "coverity":
        {
             "connect": {
                    "url": "<Connect URL>",
                        "project":{
                        "name": "<PROJECT_NAME>"
                    },
                    "stream": {
                        "name": "<STREAM_NAME>"
                    },
                    "policy": {
```

```
                    "view": "<View Name / Id>"
                }
            },
        }
    }
}
```

The above examples use the following:

- `BRIDGE_COVERITY_CONNECT_USER_NAME` and `BRIDGE_COVERITY_CONNECT_USER_PASSWORD` environment variables to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.
- `coverity.connect.url` for Coverity Connect URL.
- `coverity.connect.project.name` for project on Coverity Connect to be used.
- `coverity.connect.stream.name` for stream on Coverity Connect to be used.
- `coverity.connect.policy.view` for policy view to be used to decide if the CI pipeline should be failed or not.
- `Coverity.local` to let Synopsys bridge know if this is an on-prem Coverity Connect or a Coverity cloud deployment.

Running Coverity Connect scans using the command line

Instead of using a JSON file, you can pass arguments on the command line.

Here are the example commands that can be used with on-prem Coverity Connect:

```
export BRIDGE_COVERITY_CONNECT_USER_NAME=<COV_USER>

export BRIDGE_COVERITY_CONNECT_USER_PASSWORD=<COVERITY_PASSPHRASE>

synopsys-bridge --stage bridge \

  coverity.connect url=<COVERITY_URL> \

  coverity.connect.project.name=<COVERITY_PROJECT> \

  coverity.connect.stream.name=<COVERITY_STREAM> \

  coverity.connect.policy.view=<COVERITY_VIEW_NAME> \

  coveriy.local=true
```

Here are the example commands that can be used with Coverity cloud deployment:

```
export BRIDGE_COVERITY_CONNECT_USER_NAME=<COV_USER>

export BRIDGE_COVERITY_CONNECT_USER_PASSWORD=<COVERITY_PASSPHRASE>
```

```
synopsys-bridge --stage bridge \ coverity.connect url=<COVERITY_URL> \

   coverity.connect.project.name=<COVERITY_PROJECT> \

   coverity.connect.stream.name=<COVERITY_STREAM> \

   coverity.connect.policy.view=<COVERITY_VIEW_NAME>
```

The above examples use the following:

- `BRIDGE_COVERITY_CONNECT_USER_NAME` and `BRIDGE_COVERITY_CONNECT_USER_PASSWORD` environment variables to pass sensitive information such as password or access token to Synopsys Bridge (recommended for security purposes). Note that Synopsys Bridge automatically picks up values passed thru these environment variables.
- `--stage` argument to specify the Synopsys security product in use.
- `coverity.connect.url` for Coverity Connect URL.
- `coverity.connect.project.name` for project on Coverity Connect to be used.
- `coverity.connect.stream.name` for stream on Coverity Connect to be used.
- `coverity.connect.policy.view` for policy view to be used to decide if the CI pipeline should be failed or not.
- `Coverity.local` to let Synopsys bridge know if this is an on-prem Coverity Connect or a Coverity cloud deployment.

For the required minimum set of arguments that you need to pass to integrate Synopsys Bridge with Polaris, refer to the Polaris specific resources page under Schema Resources and Extensions *(on page)*.

For more details, see the Complete List of Synopsys Bridge Arguments *(on page 19)*.

Chapter 3. Synopsys Bridge CLI Reference

Complete List of Synopsys Bridge Arguments

This page lists all the arguments that Synopsys Bridge supports. Arguments can be passed thru environment variables, command line or a JSON file.

> **Note:**
> We recommend that you pass sensitive information such as access tokens using environment variables.

For a list of arguments that are common to all Synopsys security products, refer to Universal Synopsys Bridge Arguments *(on page 19)* below.

For product specific arguments, refer to the product specific section below:

- Polaris *(on page 20)*
- Black Duck *(on page 23)*
- Coverity Connect *(on page 29)*

Universal Synopsys Bridge Arguments

These arguments can be passed on the command line, but not as part of a JSON file.

Command	Description	Required?
`synopsys-bridge`	Command to invoke Synopsys Bridge.	Yes
`--stage`	The `--stage` command specifies a group of adapters to run (such as `--stage polaris`).	Yes
`--input`	The `--input` command loads a JSON file containing common arguments to run scans	Required for inputting a JSON files.
`--help`	Shows the help file for Synopsys Bridge.	No
`--json-log`	Outputs JSON format logs. See Logging and Diagnostics *(on page 33)*.	No
`--json-log-file`	Outputs JSON format logs in the `bridge.log` file in the Synopsys Bridge	No

Command	Description	Required?
	home directory. See Logging and Diagnostics *(on page 33)*.	
-- home	Sets a home directory.	No
-- version	Specifies a specific version of Synopsys Bridge to run.	No
-- schema	Specifies a schema to load	No
-- verbose	Turns on verbose logging.	No
`--diagnostics`	Enables debug logs under the Synopsys Bridge home directory. Creates a `diagnostics.json` file containing the final state data inside the Synopsys Bridge home directory, but masking sensitive information like tokens and passwords. See Logging and Diagnostics *(on page 33)*.	No

Polaris

Arguments to Pass

Argument	Input Mode			Required	Notes
	Command Line Argument	**Environment Variable**	**JSON field**		
Access token	`polaris.accesstoken`	`BRIDGE_POLARIS_ACCESSTOKEN`	`polaris.accesstoken`	Yes	For security reasons, it is recommended that you pass this as an environment variable.
Server URL	`polaris.serverurl`	`BRIDGE_POLARIS_SERVERURL`	`polaris.serverurl`	Yes	For security reasons, it is recommended that you

Argument	Input Mode			Required	Notes
					pass this as an environment variable.
Application Name	`polaris.application.name`	`BRIDGE_POLARIS_APPLICATION_NAME`	`polaris.application.name`	Yes	Application must be created on Polaris, and have right entitlements.
Project Name	`polaris.project.name`	`BRIDGE_POLARIS_PROJECT_NAME`	`polaris.project.name`	Yes.	if `polaris.onboarding` is set to `true`, Bridge will create the project as necessary.
Assessment Type	`polaris.assessment.types`	`BRIDGE_POLARIS_ASSESSMENT_TYPES`	`polaris.assessment.types`	Yes	Comma separated value. Accepted values `SAST` or `SCA` or `SAST,SCA`.
Tool Install Directory	`tool.install.directory`	`BRIDGE_TOOL_INSTALL_DIRECTORY`	`tool.install.directory`	No	Directory to which Bridge downloads the underlying scan tools. Defaults to `<User>/.bridge`.
Auto Create Projects	`polaris.onboarding`	`BRIDGE_POLARIS_ONBOARDING`	`polaris.onboarding`	No	If set to `true`, Bridge will attempt to create the project on Polaris if it does not ex-

Argument	Input Mode		Required	Notes
				ist. Default is `false`. **Note:** The specified application must exist on Polaris along with appropriate entitlements. Bridge will otherwise error out.
Polaris Triage	`polaris.triage`	`BRIDGE_POLARIS_TRIAGE`	`polaris.triage`	No	If you are entitled to the Auto-Triage feature on Polaris, you can use this op-

Argument	Input Mode	Required	Notes
			tion to enable the feature. Possible values are `REQUIRED`, `NOT_REQUIRED` and `NOT_ENTITLED`.

JSON Input

Here is a sample `input.json` file that can be used with Polaris:

```
{
    "data": {
        "polaris": {
            "application": {
                "name": "<Application Name>"
            },
            "project": {
                "name": "<Project Name>"
            },
            "assessment": {
                "types":  ["SCA", "SAST"]


            },
            "serverUrl": "<Polaris URL>"
        }
    }
}
```

Here are the commands to run:

```
export BRIDGE_POLARIS_ACCESSTOKEN=<POLARIS_ACCESSTOKEN>


synopsys-bridge --stage Polaris --input input.json
```

Black Duck

The base command to run the scan:

```
synopsys-bridge --stage blackduck
```

Arguments to Pass

Argument	Input Mode			Required	Notes
	Command Line Argument	Environment Variable	JSON field		
URL	`blackduck.url`	`BRIDGE_BLACKDUCK_URL`	`blackduck.url`	Yes	Black Duck URL
Token	`blackduck.token`	`BRIDGE_BLACKDUCK_TOKEN`	`blackduck.token`	Yes	Black Duck Access token
Full scan	`blackduck.scan.full`	`BRIDGE_BLACKDUCK_SCAN_FULL`	`blackduck.scan.full`	No	Performs a full/intelligent scan when set to `true`. Required and used for scanning based on SCM push events. Performs a rapid scan when set to `false` . Required for SCM pull request events. `true` or `false`. (Default: `false`).
Install Directory	`blackduck.install.directory`	`BRIDGE_BLACKDUCK_INSTALL_DIRECTORY`	`blackduck.install.directory`	No	Path to directory where `detect.jar` resides. Default: `<$HOME>/.bridge/blackduck`

Argument	Input Mode			Required	Notes
Failure severities	`blackduck.scan.failure.severities`	`BRIDGE_BLACKDUCK_SCAN_FAILURE_SEVERITIES`	`blackduck.scan.failure.severities`	No	Used by Bridge to determine whether to break the build or not. If provided, Bridge will break the build and returns exit code.
Create fix pull requests	`blackduck.automation.fixpr`	`BRIDGE_BLACKDUCK_AUTOMATION_FIXPR`	`blackduck.automation.fixpr`	No	If set to `true`, Bridge creates fix pull requests for vulnerable direct dependencies. (Default: `false`) Note: Currently only NPM is supports. Note: Requires SCM information including token as documented in section SCM Information needed for “Creating Fix Pull Requests” feature below.
Add comments to pull requests	`blackduck.automation.prcomments`	`BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENTS`	`blackduck.automation.prcomments`	No	If set to `true` , Bridge adds comments to pull requests for new is-

Argument	Input Mode	Required	Notes
			sues introduced in the pull request. Requires Rapid Scan to be run on pull require events. This flag is ignored if full scan is run. Note: Requires SCM information including token as documented in section SCM Information needed for "Adding Comments to Pull Requests" feature below.

SCM Information needed for “Creating Fix Pull Requests” feature

To use this feature, you must pass the following SCM arguments.

SCM	Argument	Input Mode			Required
		Command Line Argument	**Environment Variable**	**JSON Field**	
GitHub	User Token	`github.user.token`	`BRIDGE_GITHUB_USER_TOKEN`	`github.user.token`	Yes
	Repository Name	`github.repository.name`	`BRIDGE_GITHUB_REPOSITORY_NAME`	`github.repository.name`	Yes
	Branch Name	`github.repository.branch.name`	`BRIDGE_GITHUB_REPOSITORY_BRANCH_NAME`	github.repository.branch.name	Yes
	Repository Owner	`github.repository.owner.name`	`BRIDGE_GITHUB_REPOSITORY_OWNER_NAME`	`github.repository.owner.name`	Yes

SCM	Argument	Input Mode			Required
GitLab	GitLab API URL	`gitlab.api.url`	`BRIDGE_GITLAB_URL`	`gitlab.api.url`	Yes
	User Token	`gitlab.user.token`	`BRIDGE_GITLAB_USER_TOKEN`	`gitlab.user.token`	Yes
	Repository Name	`gitlab.repository-.name`	`BRIDGE_GITLAB_REPOSITO-RY_NAME`	`gitlab.repository-.name`	Yes
	Branch Name	`gitlab.repository-.branch.name`	`BRIDGE_GITLAB_REPOSITO-RY_BRANCH_NAME`	`gitlab.repository-.branch.name`	Yes
Azure	Azure API URL	`azure.api.url`	`BRIDGE_AZURE_API_URL`	`azure.api.url`	Yes
	User Token	`azure.user.token`	`BRIDGE_AZURE_USER_TOKEN`	`azure.user.token`	Yes
	Organization Name	`azure.organization-.name`	`BRIDGE_AZURE_ORGANIZA-TION_NAME`	`azure.organization-.name`	Yes
	Project Name	`azure.project.name`	`BRIDGE_AZURE_PROJECT_-NAME`	`azure.project.name`	Yes
	Repository Name	`azure.repository-.name`	`BRIDGE_AZURE_REPOSITO-RY_NAME`	`azure.repository-.name`	Yes
	Branch Name	`azure.repository-.branch.name`	`BRIDGE_AZURE_REPOSITO-RY_BRANCH_NAME`	`azure.repository-.branch.name`	Yes
	Pull Request Number	`azure.repository-.pull.number`	`BRIDGE_AZURE_REPOSITO-RY_PULL_NUMBER`	`azure.repository-.pull.number`	Yes

SCM Information needed for “Adding Comments to Pull Requests” feature

SCM	Argument	Input Mode			Required
		Command Line Argument	**Environment Variable**	**JSON Field**	
GitHub	User Token	`github.user.token`	`BRIDGE_GITHUB_USER_TOKEN`	`github.user.token`	Yes
	Repository Name	`github.repository-.name`	`BRIDGE_GITHUB_REPOSITO-RY_NAME`		Yes

SCM	Argument	Input Mode			Required
	Branch Name	`github.repository.branch.name`	`BRIDGE_GITHUB_REPOSITORY_BRANCH_NAME`	`github.repository.branch.name`	Yes
	Repoository Owner	`github.repository.owner.name`	`BRIDGE_GITHUB_REPOSITORY_OWNER_NAME`	`github.repository.owner.name`	Yes
	Pull Request Number	`github.repository.pull.number`	`BRIDGE_GITHUB_REPOSITORY_PULL_NUMBER`	`github.repository.pull.number`	Yes
GitLab	GitLab API URL	`gitlab.api.url`	`BRIDGE_GITLAB_URL`	`gitlab.api.url`	Yes
	User Token	`gitlab.user.token`	`BRIDGE_GITLAB_USER_TOKEN`	`gitlab.user.token`	Yes
	Repository Name	`gitlab.repository.name`	`BRIDGE_GITLAB_REPOSITORY_NAME`	`gitlab.repository.name`	Yes
	Branch Name	`gitlab.repository.branch.name`	`BRIDGE_GITLAB_REPOSITORY_BRANCH_NAME`	`gitlab.repository.branch.name`	Yes
	Pull Request Number	`gitlab.repository.pull.number`	`BRIDGE_GITLAB_REPOSITORY_PULL_NUMBER`	`gitlab.repository.pull.number`	Yes
Azure	Azure API URL	`azure.api.url`	`BRIDGE_AZURE_API_URL`	`azure.api.url`	Yes
	User Token	`azure.user.token`	`BRIDGE_AZURE_USER_TOKEN`	`azure.user.token`	Yes
	Organization Name	`azure.organization.name`	`BRIDGE_AZURE_ORGANIZATION_NAME`	`azure.organization.name`	Yes
	Project Name	`azure.project.name`	`BRIDGE_AZURE_PROJECT_NAME`	`azure.project.name`	Yes
	Repository Name	`azure.repository.name`	`BRIDGE_AZURE_REPOSITORY_NAME`	`azure.repository.name`	Yes
	Branch Name	`azure.repository.branch.name`	`BRIDGE_AZURE_REPOSITORY_BRANCH_NAME`	`azure.repository.branch.name`	Yes

JSON Input

Here is a sample `input.json` file that can be used with Black Duck:

```
{
    "data": {
        "blackduck": {
            "url": <BlackDuck url>,
            "scan": {
                "full": true,
                "failure": {
                    "severities": ["CRITICAL"]
                }
            }
        }
    }
}
```

Here are the commands to run:

```
export BRIDGE_BLACKDUCK_TOKEN=<BLACKDUCK_TOKEN>

synopsys-bridge --stage blackduck --input input.json
```

Coverity Connect

Argument	Input Mode			Required	Notes
	Command Line Argument	**Environment Variable**	**JSON field**		
Coverity URL	`coverity.connect.url`	`BRIDGE_COVERITY_CONNECT_URL`	`coverity.connect.url`	Yes	
User Name	`coverity.connect.user.name`	`BRIDGE_COVERITY_CONNECT_USER_NAME`	`coverity.connect.user.name`	Yes	For security reasons it is recommended to pass this as an environmental variable.
Password	`coverity.connect.`	`BRIDGE_COVERITY_CON`	`coverity.connect.`	Yes	For security reasons it is recommended to pass this as an environmental variable.

Argument	Input Mode			Required	Notes
	`.user-.password`	`NECT_USER_-PASSWORD`	`.user-.password`		
Project Name	`coveri-ty.con-nect-.project-.name`	`BRIDGE_-COVERITY_-CONNECT_-PROJECT_-NAME`	`coveri-ty.con-nect-.project-.name`	Yes	Project must exist on Coverity Instance
Stream Name	coveri-ty.connec-t.stream-.name	`BRIDGE_-COVERITY_-CONNECT_-STREAM_NAME`	`coveri-ty.con-nect-.stream-.name`	Yes	Stream must exist on Coverity Instance.
View	`coveri-ty.con-nect.pol-icy.view`	`BRIDGE_COV-ERITY_CON-NECT_POLI-CY_VIEW`	`coveri-ty.con-nect.pol-icy.view`	No	Coverity platform's view name/ID. Bridge will break the build if issues are found in the view provided by user and returns exit code *(on page 33)*.
Add comments to pull requests	`coveri-ty.con-nect.au-tomation-.prcom-ment`	`BRIDGE_COV-ERITY_CON-NECT_AU-TOMATION_-PRCOMMENT`	`coveri-ty.con-nect.au-tomation-.prcom-ment`	No	If set to `true`, Bridge adds comments to pull requests for new issues introduced in the pull request. Requires Rapid Scan to be run on pull require events. This flag is ignored if full scan is run. Note: Requires SCM information including token as documented in section SCM Information needed for “Adding Comments to Pull Requests” feature.
Install directory	`cover-ity.in-stall.di-rectory`	`BRIDGE_COV-ERITY_INS-TALL_DIREC-TORY`	`cover-ity.in-stall.di-rectory`	No	Path to directory where `coverity` resides. Default: `<$HOME>/.bridge/coverity`.

Argument	Input Mode			Required	Notes
local analysis	`coverity.local`	`BRIDGE_COVERITY_LOCAL`	`coverity.local`	No	To use Synopsys Bridge with on-prem Coverity Connect, set this to `true`. When set to true, Bridge will download full analysis kit and will perform capture and analysis locally. With Coverity cloud deployments, Synopsys uses Thin Client and this option should be set to `false`. Default: `false`.

Note:
To use Synopsys Bridge with on-prem Coverity Connect, you must set the “Coverity.local” to true as described above.

Here is a sample `input.json` file that can be used with Coverity Cloud:

:

```
{
    "data":
    {
        "coverity":
        {
            "connect": {
                "url": "<Connect URL>",


                "project":{
                    "name": "<PROJECT_NAME>"
                },
                "stream": {
                  "name": "<STREAM_NAME>"
                },
                "policy": {
                    "view": "<View Name / Id>"
```

```
                },
                "automation": {
                    "prcomment" : false
                }

            }
        }
    }
}
```

Here is a sample `input.json` file that can be used with on-prem Coverity Connect:

```
{
    "data":
    {
        "coverity":
        {
            "connect": {
                "url": "<Connect URL>",

                "project":{
                    "name": "<PROJECT_NAME>"
                },
                "stream": {
                  "name": "<STREAM_NAME>"
                },
                "policy": {
                    "view": "<View Name / Id>"
                },
                "automation": {
                    "prcomment" : false
                }

            },
            “local” : true}
        }
    }
}
```

Here are the commands to run:

```
export BRIDGE_COVERITY_CONNECT_USER_NAME=<COV_USER>

export BRIDGE_COVERITY_CONNECT_USER_PASSWORD=<COVERITY_PASSPHRASE>

synopsys-bridge --stage blackduck --input input.json
```

Exit Codes

After running a Synopsys Bridge command, you will receive a response code (see below) while full response details appear in the console. If Synopsys Bridge runs into problems, it outputs colored `ERROR` and `WARN` lines in the console response.

Synopsys Bridge replies with different exit codes depending upon execution results. Any exit code other than `0` should be seen as a build-breaking condition in your CI/CD platform.

Code	Code Name	Description
`0`	Normal	Synopsys Bridge exited without any errors.
`1`	UndefinedError	Undefined errors. Review the log file for details.
`2`	AdapterError	Synopsys Bridge received a non-0 exit code from an internal adapter. Review the log file for details.
`3`	ShutdownFailed	Synopsys Bridge failed to shut itself down after running the command. Review the log for details.
`8`	BridgeBuildBreak	The config option `bridge.break` is set to true but Synopsys Bridge is unable to enforce this. As a workaround, create a simple script to call Synopsys Bridge and implement build break logic in your script.
`9`	StartupFailed	Failed to initiate Synopsys Bridge. Review the log for details.

Logging and Diagnostics

Synopsys Bridge offers multiple logging and diagnostic options. By default, logs are written to `<current_working_directory>/.bridge` directory. User can change this default location by passing the `--home <directory_path>` option.

Logging

Synopsys Bridge offers multiple logging options.

- Pass `--json-log` to output JSON format logs.
- Pass `--json-log-file` to enable JSON format logs in the `bridge.log` file in the Synopsys Bridge home directory .

Diagnostics

To enable Synopsys Bridge diagnostics mode, pass a `--diagnostics` command line option. With this option set, Synopsys Bridge:

- writes additional diagnostics information to `bridge.log`.
- passes diagnostics related options to underlying tools so that they create logs under the Synopsys Bridge home directory.
- writes execution state date to `diagnostics.json` file under the Synopsys Bridge home directory.

Chapter 4. GitHub - Synopsys Action

The Synopsys GitHub Action can be used to integrate Synopsys security testing into your CI pipeline. You can download Synopsys GitHub Action directly from the GitHub Marketplace link at: https://github.com/marketplace/actions/synopsys-action.

By including and configuring the Synopsys Action in your *workflow*.yml file, you can quickly integrate Synopsys security products into your CI pipeline. We recommend using GitHub secrets for sensitive data like access tokens

For more information, see:

GitHub Prerequisites

Before configuring Synopsys Action into your workflow, you must meet the following prerequisites:

GitHub Runner Setup

- Runners are the machines that execute jobs in a GitHub Actions workflow. To use GitHub runners in your project, GitHub Actions must be enabled for a repository/organization settings in order for required workflows to run (**Repository Settings** → **SelectActions** → **General** → **Actions permissions**).
- GitHub runner can be Self-hosted or GitHub-hosted. For installing Self-hosted runners, see Self-hosted runners. For installing GitHub-hosted runners, see GitHub-hosted runners.

Configure GitHub Secrets

Sensitive data such as access tokens, user names, passwords and even URLs must be configured using GitHub secrets (**GitHub** → **Project** → **Settings** → **Secrets and Variables** → **Actions**).

Configure GitHub Token

`github_token` is required as input when running Black Duck Fix PR, Black Duck/Coverity PR Comment. There are two different types of tokens that can be passed to `github_token`:

- Token can be GitHub specified `secrets.GITHUB_TOKEN` with required workflow read and write permissions (**GitHub** → **Project** → **Settings** → **Actions** → **General** → **Workflow Permissions**). It will be created by GitHub at start of each workflow run.
- If you need a token that requires permissions that aren't available in the `secrets.GITHUB_TOKEN`, create a Personal Access Token (PAT) with required scopes (**Select Profile Photo** → **Settings** → **Developer Settings** → **Personal access tokens**). For more information, see Granting Additional Permissions. PAT must have repo and api scope to perform Black Duck Fix PR or Black Duck/ Coverity PR Comment.

Create workflow

Create a new workflow (**GitHub** → **Project** → **Actions** → **New Workflow** → **Setup a workflow yourself**) and configure he required fields. Push those changes and GitHub runner will initiate the workflow which can be seen on the **Actions** tab on main page of the repository.

Using Synopsys GitHub Action for Polaris

Before running a pipeline using the Synopsys GitHub Action with Polaris, you must set the appropriate applications and entitlements in your Polaris environment.

Using Synopsys Action, you can perform scans on push events to main branches. Pull request scanning is currently not supported for Polaris.

Add the following code block to your existing *`workflow`*`.yml` file in your `.github/workflows` directory. (If you need to create a workflow, go to the repository you're integrating with Polaris on the GitHub UI, click the **Actions** tab at the top, then click **New Workflow**.)

Below is an example of a *`workflow`*`.yml` file configured for Polaris.

```
name: polaris-sig-action

on:
  push:
    branches: [ main, master, develop, stage, release ]
  workflow_dispatch:
jobs:
  build:
    runs-on: [ ubuntu-latest ]
    steps:
      - name: Checkout Source
        uses: actions/checkout@v3
```

```
- name: Polaris Scan
  uses: synopsys-sig/synopsys-action@v1.2.0
  with:
    polaris_serverUrl: ${{ secrets.POLARIS_SERVERURL }}
    polaris_accessToken: ${{ secrets.POLARIS_ACCESSTOKEN }}
    polaris_application_name: ${{ github.event.repository.name }}
    polaris_project_name: ${{ github.event.repository.name }}
    ### Accepts Multiple Values
    polaris_assessment_types: "SAST,SCA"
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # include_diagnostics: true
```

Table 1. List of mandatory and optional parameters for Polaris

Input Parameter	Description	Mandatory / Optional
`polaris_serverUrl`	Polaris URL	Mandatory
`polaris_accessToken`	Polaris Access token	Mandatory
`polaris_application_name`	Polaris Application name	Mandatory
`polaris_project_name`	Polaris Project name	Mandatory
`polaris_assessment_types`	Polaris assessment types. Example: SCA or SAST or SAST,SCA	Mandatory

Using Synopsys GitHub Action for Black Duck

The Synopsys Action supports both self-hosted (e.g. on-prem) and Synopsys-hosted Black Duck Hub instances.

In the default Black Duck Hub permission model, projects and project versions are created on the fly and as needed. Ensure that permissions needed to create projects and project versions are granted on Black Duck Hub.

Synopsys action runs full “intelligent” Black Duck scans on SCM push events and “rapid” ephemeral scans for SCM pull request events as shown in the example below.

Note:
Detect specific options can be passed to Synopsys Bridge thru Detect environment variables.

Below is an example of a `workflow.yml` file configured for Black Duck.

```
name: bd-sig-action


on:
  push:
    branches: [ main, master, develop, stage, release ]
  pull_request:
    branches: [ main, master, develop, stage, release ]
  workflow_dispatch:
jobs:
  build:
    runs-on: [ ubuntu-latest ]
    steps:
      - name: Checkout Source
        uses: actions/checkout@v3


      - name: Black Duck Full Scan
        if: ${{ github.event_name != 'pull_request' }}
        uses: synopsys-sig/synopsys-action@v1.2.0
        ### Use below configuration to set specific detect environment variables
        env:
          DETECT_PROJECT_NAME: ${{ github.event.repository.name }}
        with:
          blackduck_url: ${{ secrets.BLACKDUCK_URL }}
          blackduck_apiToken: ${{ secrets.BLACKDUCK_API_TOKEN }}
          blackduck_scan_full: true
          ### Accepts Multiple Values
          blackduck_scan_failure_severities: 'BLOCKER,CRITICAL'
          ### Uncomment below configuration to enable automatic fix pull request creation if vulnerabilities are
 reported
          # blackduck_automation_fixpr: true
          # github_token: ${{ secrets.GITHUB_TOKEN }} # Mandatory when blackduck_automation_fixpr is set to 'true'
          ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
```

```
      # include_diagnostics: true


  - name: Black Duck PR Scan
    if: ${{ github.event_name == 'pull_request' }}
    uses: synopsys-sig/synopsys-action@v1.2.0
    ### Use below configuration to set specific detect environment variables
    env:
      DETECT_PROJECT_NAME: ${{ github.event.repository.name }}
    with:
      blackduck_url: ${{ secrets.BLACKDUCK_URL }}
      blackduck_apiToken: ${{ secrets.BLACKDUCK_API_TOKEN }}
      blackduck_scan_full: false
      ### Below configuration is used to enable automatic pull request comment based on Black Duck scan result
      blackduck_automation_prcomment: true
      github_token: ${{ secrets.GITHUB_TOKEN }} # Mandatory when blackduck_automation_prcomment is set to 'true'
      ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
      # include_diagnostics: true
```

Setting Fix Pull requests creation works as follows:

- **`blackduck_automation_fixpr`**: By default, fix pull request creation is disabled (Synopsys Action will not create fix pull requests for vulnerable direct dependencies.). To enable this feature, set `blackduck_automation_fixpr` as `true`.
- **`github_token`**: You must pass `github_token` parameter with required permissions. The token can be GitHub `secrets.GITHUB_TOKEN` with required permissions. For more information on GitHub tokens see the GitHub documentation
- Due to rate limit restriction of GitHub rest API calls, note that GitHub might limit the number of pull requests that are created by Synopsys Action.

Table 2. List of mandatory and optional parameters for Black Duck

Input Parameter	Description	Mandatory / Optional
`blackduck_url`	Black Duck URL	Mandatory
`blackduck_apiToken`	Black Duck API token	Mandatory
`blackduck_install_directory`	Installation directory for Black Duck	Optional

Table 2. List of mandatory and optional parameters for Black Duck (continued)

Input Parameter	Description	Mandatory / Optional
`blackduck_scan_full`	Specifies whether full scan is required or not. Full "intelligent" scan is to be used for push events and rapid scan for pull request events. Supported values: true or false	Optional
`blackduck_scan_failure_severities`	Black Duck scan failure severities. Supported values: ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED	Optional
`blackduck_automation_prcomment`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Black Duck PR Comment. Default: false	Optional
`blackduck_automation_fixpr`	Flag to enable automatic creation for fix pull requests for vulnerable direct dependencies. Default: false	Optional

Table 2. List of mandatory and optional parameters for Black Duck (continued)

Input Parameter	Description	Mandatory / Optional
	Black Duck automation fix pull request is currently supported for NPM projects only.	
`github_token`	GitHub Access Token Example: `github_token: ${{ secrets.GITHUB_TOKEN }}`	Mandatory if `blackduck_automation_fixpr` or `blackduck_automation_prcomment` is set as true

Using Synopsys GitHub Action for Coverity Cloud Deployment with Thin Client

Synopsys GitHub Action only supports the Kubernetes-based Coverity cloud deployment model, which uses a small footprint thin client to capture the source code and submit an analysis job running on the server. This removes the need for a multi-gigabyte software installation in your GitHub Runner.

On push events, a full Coverity scan will be run and results are committed to the Coverity Connect database.

On pull request events, comments are added to pull requests for new issues found by the scan if `coverity_automation_prcomment` is set to `true` (see example below). Note that scan results are not committed to Coverity Connect database in this case.

Before running the pipeline with Synopsys Action, make sure the specified project and stream exist in your Coverity Connect server environment.

Below is an example of a *`workflow`*`.yml` file configured for Coverity Cloud Deployment.

```
name: cnc-sig-action

on:

  push:

    branches: [ main, master, develop, stage, release ]

  pull_request:

    branches: [ main, master, develop, stage, release ]

  workflow_dispatch:

jobs:
```

```
build:
  runs-on: [ ubuntu-latest ]
  steps:
    - name: Checkout Source
      uses: actions/checkout@v3


    - name: Coverity Full Scan
      if: ${{ github.event_name != 'pull_request' }}
      uses: synopsys-sig/synopsys-action@v1.2.0
      with:
        coverity_url: ${{ secrets.COVERITY_URL }}
        coverity_user: ${{ secrets.COVERITY_USER }}
        coverity_passphrase: ${{ secrets.COVERITY_PASSPHRASE }}
        coverity_project_name: ${{ github.event.repository.name }}
        coverity_stream_name: ${{ github.event.repository.name }}-${{ github.ref_name }}
        coverity_policy_view: 'Outstanding Issues'
        ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
        # include_diagnostics: true


    - name: Coverity PR Scan
      if: ${{ github.event_name == 'pull_request' }}
      uses: synopsys-sig/synopsys-action@v1.2.0
      with:
        coverity_url: ${{ secrets.COVERITY_URL }}
        coverity_user: ${{ secrets.COVERITY_USER }}
        coverity_passphrase: ${{ secrets.COVERITY_PASSPHRASE }}
        coverity_project_name: ${{ github.event.repository.name }}
        coverity_stream_name: ${{ github.event.repository.name }}-${{ github.base_ref }}
        ### Below configuration is used to enable feedback from Coverity security testing as pull request comment
        coverity_automation_prcomment: true
        github_token: ${{ secrets.GITHUB_TOKEN }} # Mandatory when coverity_automation_prcomment is set to 'true'
        ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
        # include_diagnostics: true
```

Table 3. List of mandatory and optional parameters for Coverity

Input Parameter	Description	Mandatory / Optional
`coverity_url`	Coverity URL	Mandatory
`coverity_user`	Coverity username	Mandatory
`coverity_passphrase`	Coverity passphrase	Mandatory
`coverity_project_name`	Coverity project name. Tip: Many customers prefer to set their Coverity project and stream names to match the GitHub repository name	Mandatory
`coverity_stream_name`	Coverity stream name	Mandatory
`coverity_install_directory`	Installation directory of Coverity	Optional
`coverity_policy_view`	ID or name of policy view to be used to enforce the “break the build” policy. If issues are found in the specified this view, build will be failed. Example: `coverity_policy_view: '100001'` or `coverity_policy_view: 'Outstanding Issues'`	Optional
`coverity_automation_prcomment`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Coverity PR Comment.	Optional

Table 3. List of mandatory and optional parameters for Coverity (continued)

Input Parameter	Description	Mandatory / Optional
	Default: false	
`github_token`	GitHub Access Token Example: `github_token: ${{ secrets.GITHUB_TOKEN }}`	Mandatory if coverity_automation_prcomment is set as true

Additional GitHub Configuration

The following parameters can be used for Polaris, Black Duck or Coverity Connect.

- `synopsys_bridge_path`: Provides the path to Synopsys Bridge.

  Note:
  If this is not explicitly specified, then the integration defaults to `$HOME/synopsys-bridge`. If the installed version of Synopsys Bridge is not the latest, then the latest version of Synopsys Bridge is downloaded unless you specify the version to use explicitly (as documented below).

- `bridge_download_url`: Specifies the URL to the Synopsys Bridge zip file to be downloaded and used.

  Note:
  If `bridge_download_url` is not provided, Synopsys GitHub Action downloads the latest version of Synopsys Bridge from the default SIG-REPO download location.

- `bridge_download_version`: Specifies the Synopsys Bridge version to use. If provided, the specified version of Synopsys Bridge will be automatically downloaded and used. If not, the latest version is downloaded and used.
- `include_diagnostics`: When set to `true`, Synopsys Bridge diagnostic files are created and posted to GitHub. Additionally, diagnostics_retention_days can be used to specify the number of days the diagnostics files are retained for. Default value is 90. Accepted range of values is from 1 to 90.

Chapter 5. GitLab – Synopsys Template

Synopsys GitLab Template allows you to configure your GitLab pipeline to run Synopsys security testing and act on the results.

Synopsys GitLab Template Marketplace link is https://gitlab.com/synopsys/synopsys-template.

Additional information

For additional GitLab integration information, see:

GitLab Prerequisites

Before configuring Synopsys Template into your GitLab pipeline, set up the following.

GitLab Runner Setup

- GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline. To use GitLab Runner in your project, you must have the maintainer or owner role for the project.
- A GitLab runner can be self-managed or SaaS runners managed by GitLab.
- A GitLab self-managed runner can be installed and used on GNU/Linux, macOS and Windows. For more details refer: Install GitLab Runner
- To set up project specific self-managed runner, go to (**Project Settings** → **CI/CD** → **Runners**) and configure.
- During runner registration, choose executor as `shell`.
- Make sure you have curl and unzip package tools installed in self-managed/SaaS runner (Linux/Mac).
- Synopsys Template supports both Project runners and Shared runners (except Shared Mac Runners).

Configure GitLab Variables

- Sensitive data such as access tokens, user names, passwords and even URLs must be configured using GitLab variables.
- These can be added at the Project, Group or Global scopes (Global for self-managed GitLab instances only).
- To add variables, go to **Settings** → **CI/CD** → **Variables**. Be sure to mask passwords and tokens to avoid them being exposed in logs. For more details see GitLab CI/CD variables.

Configure Gitlab User Token

- `BRIDGE_GITLAB_USER_TOKEN` is required as input when running Black Duck Fix PR, Black Duck/Coverity PR Comment.
- Generate a Personal Access Token (PAT) from GitLab (**User Settings** → **Access Tokens**) and store it as secret variable or store and fetch it from vault.
- PAT must have **api** scope to perform Black Duck Fix PR or Black Duck/Coverity PR Comment. For more details, see: Personal access tokens

Create a .gitlab-ci.yml file

- Before running a pipeline using the Synopsys Template, add a `.gitlab-ci.yml` file to your project by adding an `include` entry.
- Push those changes and a GitLab runner picks up the job and initiates the pipeline.

Using the Synopsys GitLab Template with Polaris

It is recommended that you configure sensitive information such as access tokens and URLs using GitLab secrets.

Before running a pipeline using the Synopsys Template, add `.gitlab-ci.yml` to your project by adding an `include` entry, as in the example below.

```
include:
   - project: synopsys/synopsys-template
     ref: v1.1.0
     file: templates/synopsys-template.yml
  ### Use below configuration for accessing synopsys-template in GitLab self-managed
  # - remote: 'https://gitlab.com/synopsys/synopsys-template/-/raw/v1.0.0/templates/synopsys-template.yml'
variables:
  BRIDGE_POLARIS_SERVERURL: $POLARIS_SERVER_URL
```

```
  BRIDGE_POLARIS_ACCESSTOKEN: $POLARIS_ACCESS_TOKEN

  BRIDGE_POLARIS_APPLICATION_NAME: $CI_PROJECT_NAME

  BRIDGE_POLARIS_PROJECT_NAME: $CI_PROJECT_NAME

  ### Accepts Multiple Values

  BRIDGE_POLARIS_ASSESSMENT_TYPES: 'SCA,SAST'

stages:

  - polaris_scan

synopsys_template_execution:

  stage: polaris_scan

  tags:

    - linux # Name of your GitLab runner

  extends: .run-synopsys-tools # Used for bash

  # extends: .run-synopsys-tools-powershell # Used for powershell


  ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded

  # variables:

  #   INCLUDE_DIAGNOSTICS: 'true'

  # artifacts:

  #   when: always

  #   paths:

  #     - .bridge
```

Note:
Polaris does not currently support the analysis of merge requests. We recommend running the Synopsys GitLab Template on pushes to main branches.

Once you push the changes above, an active runner will pick up the job and initiate the pipeline.

Table 4. List of mandatory and optional parameters for Polaris

Input Parameter	Description	Mandatory / Optional
`BRIDGE_POLARIS_SERVERURL`	Polaris server URL	Mandatory
`BRIDGE_POLARIS_ACCESSTOKEN`	Polaris access token	Mandatory
`BRIDGE_POLARIS_APPLICATION_-NAME`	Application name in Polaris	Mandatory
`BRIDGE_POLARIS_PROJECT_NAME`	Project name in Polaris	Mandatory

Table 4. List of mandatory and optional parameters for Polaris (continued)

Input Parameter	Description	Mandatory / Optional
`BRIDGE_POLARIS_ASSESSMENT_-TYPES`	Polaris assessment types Example: SCA,SAST	Mandatory

Using the Synopsys GitLab Template with Black Duck

Synopsys GitLab Template supports both self-hosted (on-prem) and Synopsys-hosted Black Duck Hub instances.

In the default Black Duck Hub permission model, projects and project versions are created on the fly as needed. Ensure that permissions needed to create projects and project versions are granted on Black Duck Hub.

Before running a pipeline using the Synopsys GitLab Template and Black Duck, add `.gitlab-ci.yml` to your project by adding an `include` entry, as in the example below.

```
include:
  - project: synopsys/synopsys-template
    ref: v1.1.0
    file: templates/synopsys-template.yml
  ### Use below configuration for accessing synopsys-template in Gitlab self-managed
  # - remote: 'https://gitlab.com/synopsys/synopsys-template/-/raw/v1.1.0/templates/synopsys-template.yml'


stages:
  - blackduck_scan


variables:
  SCAN_BRANCHES: "/^(main|master|develop|stage|release|feature_branch)$/" # Add branches where you want to run Black
 Duck scan


synopsys_template_execution:
  stage: blackduck_scan
  variables:
    BRIDGE_BLACKDUCK_URL: $BLACKDUCK_URL
    BRIDGE_BLACKDUCK_TOKEN: $BLACKDUCK_API_TOKEN
    ### Use below configuration to set specific detect environment variables
```

```
    DETECT_PROJECT_NAME: $CI_PROJECT_NAME

    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded

  #   INCLUDE_DIAGNOSTICS: 'true'

  # artifacts:

  #   when: always

  #   paths:

  #   - .bridge

  rules:

    ### Use below configuration to run Black Duck full scan

    - if: ($CI_COMMIT_BRANCH =~ $SCAN_BRANCHES && $CI_PIPELINE_SOURCE != 'merge_request_event')

      variables:

        BRIDGE_BLACKDUCK_SCAN_FULL: 'true'

        ### Accepts Multiple Values

        BRIDGE_BLACKDUCK_SCAN_FAILURE_SEVERITIES: 'BLOCKER,CRITICAL'

        ### Uncomment below configuration to enable automatic fix pull request creation if vulnerabilities are reported

        # BRIDGE_BLACKDUCK_AUTOMATION_FIXPR: 'true'

        BRIDGE_GITLAB_USER_TOKEN: $GITLAB_USER_TOKEN # Mandatory when BRIDGE_BLACKDUCK_AUTOMATION_FIXPR is set to
'true'

    ### Use below configuration to run Black Duck PR scan

    - if: ($CI_MERGE_REQUEST_TARGET_BRANCH_NAME =~ $SCAN_BRANCHES && $CI_PIPELINE_SOURCE == 'merge_request_event')

      variables:

        BRIDGE_BLACKDUCK_SCAN_FULL: 'false'

        BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT: 'true'

        BRIDGE_GITLAB_USER_TOKEN: $GITLAB_USER_TOKEN

  tags:

    - linux # Name of your Gitlab runner

  extends: .run-synopsys-tools # Used for bash.

  #extends: .run-synopsys-tools-powershell # Used for powershell
```

Table 5. List of mandatory and optional parameters for Black Duck

Input Parameter	Description	Mandatory/Optional
`BRIDGE_BLACKDUCK_URL`	Black Duck server URL	Mandatory
`BRIDGE_BLACKDUCK_TOKEN`	Black Duck API token	Mandatory
`BRIDGE_BLACKDUCK_INSTALL__-DIRECTORY`	Installation directory for Black Duck	Optional

Table 5. List of mandatory and optional parameters for Black Duck (continued)

Input Parameter	Description	Mandatory/Optional
`BRIDGE_BLACKDUCK_SCAN_FULL`	Specifies whether full scan is required or not. Full "intelligent" scan is to be used for push events and rapid scan for pull request events.	Optional
`BRIDGE_BLACKDUCK_SCAN_FAILURE_SEVERITIES`	Black Duck scan failure severities. Supported values: ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED	Optional
`BRIDGE_BLACKDUCK_AUTOMATION_FIXPR`	Option to enable automatic creation for fix pull requests for vulnerable direct dependencies. Default: false Supported values: true or false	Optional
`BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Black Duck PR Comment. Default: false	Optional

Table 5. List of mandatory and optional parameters for Black Duck (continued)

Input Parameter	Description	Mandatory/Optional
`BRIDGE_GITLAB_USER_TOKEN`	Gitlab User Access Token Example: `BRIDGE_GITLAB_USER_TOKEN: $GITLAB_ACCESS_TOKEN`	Mandatory when `BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT` or `BRIDGE_BLACKDUCK_AUTOMATION_FIXPR` is set as `true`.

Note:
Detect specific options can be passed to Synopsys Bridge thru Detect environment variables.

Using the Synopsys GitLab Template for Coverity Cloud Deployment with Thin Client

Before running Coverity using the Synopsys Template, ensure the appropriate project and stream are set in your Coverity Connect server environment, as in the example below.

Note:
Currently,Synopsys Template only supports the Coverity cloud deployment with Thin Client.

```
include:
  - project: synopsys/synopsys-template
    ref: v1.1.0
    file: templates/synopsys-template.yml
  ### Use below configuration for accessing synopsys-template in Gitlab self-managed
  # - remote: 'https://gitlab.com/synopsys/synopsys-template/-/raw/v1.1.0/templates/synopsys-template.yml'


stages:
  - coverity_scan


variables:
  SCAN_BRANCHES: "/^(main|master|develop|stage|release|feature_branch)$/" # Add branches where you want to run Coverity
 scan


synopsys_template_execution:
  stage: coverity_scan
```

```
  variables:
    BRIDGE_COVERITY_CONNECT_URL: $COVERITY_URL
    BRIDGE_COVERITY_CONNECT_USER_NAME: $COVERITY_USER
    BRIDGE_COVERITY_CONNECT_USER_PASSWORD: $COVERITY_PASSWORD
    BRIDGE_COVERITY_CONNECT_PROJECT_NAME: $CI_PROJECT_NAME
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
  #   INCLUDE_DIAGNOSTICS: 'true'
  # artifacts:
  #   when: always
  #   paths:
  #   - .bridge
  rules:
    - if: ($CI_COMMIT_BRANCH =~ $SCAN_BRANCHES && $CI_PIPELINE_SOURCE != 'merge_request_event')
      variables:
        BRIDGE_COVERITY_CONNECT_STREAM_NAME: $CI_PROJECT_NAME-$CI_COMMIT_BRANCH
        BRIDGE_COVERITY_CONNECT_POLICY_VIEW: 'Outstanding Issues'
    ### Use below configuration to run Coverity PR scan
    - if: ($CI_MERGE_REQUEST_TARGET_BRANCH_NAME =~ $SCAN_BRANCHES && $CI_PIPELINE_SOURCE == 'merge_request_event')
      variables:
        BRIDGE_COVERITY_CONNECT_STREAM_NAME: $CI_PROJECT_NAME-$CI_MERGE_REQUEST_TARGET_BRANCH_NAME
        ### Below configuration is used to enable feedback from Coverity security testing as pull request comment
        BRIDGE_COVERITY_AUTOMATION_PRCOMMENT: 'true'
        BRIDGE_GITLAB_USER_TOKEN: $GITLAB_USER_TOKEN # Mandatory when BRIDGE_COVERITY_AUTOMATION_PRCOMMENT is set to
'true'
  tags:
    - linux # Name of your Gitlab runner
  extends: .run-synopsys-tools # Used for bash.
  #extends: .run-synopsys-tools-powershell # Used for powershell
```

Table 6. List of mandatory and optional parameters for Coverity cloud

Input Parameter	Description	Mandatory/Optional
`BRIDGE_COVERITY_CONNECT_URL`	Coverity server URL	Mandatory
`BRIDGE_COVERITY_CONNECT_USER_-NAME`	Coverity username	Mandatory
`BRIDGE_COVERITY_CONNECT_USER_-PASSWORD`	Coverity passphrase	Mandatory

Table 6. List of mandatory and optional parameters for Coverity cloud (continued)

Input Parameter	Description	Mandatory/Optional
`BRIDGE_COVERITY_CONNECT_-PROJECT_NAME`	Project name in Coverity	Mandatory
`BRIDGE_COVERITY_CONNECT_-STREAM_NAME`	Stream name in Coverity	Mandatory
`BRIDGE_COVERITY_INSTALL_DIREC-TORY`	Installation directory of Coverity	Optional
`BRIDGE_COVERITY_CONNECT_POLI-CY_VIEW`	ID or name of policy view to be used to enforce the “break the build” policy. If issues are found in the specified this view, build will be failed. Example: `coverity_policy_view: '100001'` or `coverity_policy_-view: 'Outstanding Issues'`	Optional
`BRIDGE_COVERITY_AUTOMATION_-PRCOMMENT`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Coverity PR Comment. Default: false	Optional
`BRIDGE_GITLAB_USER_TOKEN`	Gitlab User Access Token Example: `BRIDGE_GITLAB_USER_TO-KEN: $GITLAB_USER_TOKEN`	Mandatory when `BRIDGE_COVERI-TY_AUTOMATION_PRCOMMENT` is set as `true`.

Additional GitLab Configuration

The following optional parameters can be used for Polaris, Black Duck or Coverity Connect.

- `SYNOPSYS_BRIDGE_PATH`: Provide a path, where you want to configure or already configured Synopsys Bridge. Optional.

  Note:
  If this is not explicitly specified, then the integration defaults to `$HOME/synopsys-bridge`.
  If the installed version of Synopsys Bridge is not the latest, then the latest version of Synopsys Bridge is downloaded unless you specify the version to use explicitly (as documented below).

- `DOWNLOAD_BRIDGE_URL`: Use this to specify the URL to the Synopsys Bridge zip file to be downloaded from and used.
- `DOWNLOAD_BRIDGE_VERSION`: Use this to specify the Synopsys Bridge version to use. If provided, the specified version of Synopsys Bridge will be automatically downloaded and used. If not, the latest version is downloaded and used.

  Note:
  If `DOWNLOAD_BRIDGE_URL` is not provided, Synopsys GitHub Action downloads the latest version of Synopsys Bridge from the default SIG-REPO download location.

- `INCLUDE_DIAGNOSTICS`: When set to `true`, Synopsys Bridge diagnostic files are created.

  Note:
  While including Synopsys Bridge diagnostic files, default expiry time for uploaded artifacts is 30 days. Refer to SCM documentation for more details :https://docs.gitlab.com/ee/ci/jobs/job_artifacts.html.

Chapter 6. Azure DevOps - Synopsys Security Scan

Synopsys Security Scan Extension for Azure DevOps enables you to integrate Synopsys security testing intoyour Azure pipeline.

The marketplace link for Synopsys Security Scan for Azure DevOps Visual Studio is https://marketplace.visualstudio.com/items?itemName=synopsys-security-scan.synopsys-security-scan.

Additional Info

For additional Azure integration information, see:

- Azure Prerequisites *(on page 55)*
- Using Azure DevOps Extension with Polaris *(on page 56)*
- Using Azure DevOps Extension with Black Duck *(on page 57)*
- Using Azure DevOps Extension with Coverity Connect with Thin Client *(on page 60)*
- Additional Azure Configuration *(on page 54)*

Azure Prerequisites

Before adding Synopsys Security Scan in your azure pipeline, note the following prerequisites:

Azure Agent Setup

Azure agents are required and can be installed and used on GNU/Linux, macOS, Windows and Docker. See https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser for details. You can use Microsoft-hosted agents as well to scan your code using Azure Pipelines.

Configure Variables

Sensitive data such as access tokens, user names, passwords and even URLs must be configured using variable groups (**Project** → **Pipelines** → **Library** → **New Variable Group**).

`AZURE_TOKEN` is required as input when running Black Duck Fix PR, Black Duck/Coverity PR Comment. There are two different types of tokens to pass to `AZURE_TOKEN`:

- To use `AZURE_TOKEN: $(System.AccessToken)`, you must enable this in the Azure interface. Go to **Project** → **Project Settings** → **Repository** → **Security** → **Build Service** and set **Contribute to pull requests**, **Create branch** and **Delete or disable repository** to **Allow**. Confirm `System.AccessToken`

has **Contribute to PR** permissions (**Project → Project Settings → Repositories → Security → Build Service User**).

- To use `AZURE_TOKEN: $(PAT_TOKEN)`, PAT token should have minimum permissions **Code - Full** and **Pull Request Threads - Read & write**. See Use personal access tokens for more details.

f you like Synopsys Security Scan to add comments to pull requests (supported forBlack Duck and Coverity), enable **Buildvalidation policy** (**Project→ProjectSettings → Repositories→Branch Policy →Add branch protection**) to trigger the pipeline on any PR or push event to a branch (usually main or master branch). See Build Validation for more details.

Configure Azure Pipeline

Create a new pipeline or use existing pipeline (**Project → Pipelines → New Pipeline**) and configure required fields. Push those changes and agent will pick up the job and initiate the pipeline.

Using Azure DevOps Extension with Polaris

Before running a pipeline using the Synopsys Security Scan and Polaris, add `azure-pipelines.yml` to your project. Configure sensitive data such as usernames, passwords and URLs using pipeline variables. Push the changes and an agent will pick up the job and initiate the pipeline. Here is an example `azure-pipelines.yml` that you can use with Polaris:

```
trigger:
- main


pool:
  vmImage: ubuntu-latest


variables:
  - group: polaris


steps:
- task: SynopsysSecurityScan@1.0.0
  displayName: 'Polaris Scan'
  inputs:
    BRIDGE_POLARIS_SERVERURL: $(POLARIS_SERVER_URL)
    BRIDGE_POLARIS_ACCESSTOKEN: $(POLARIS_ACCESS_TOKEN)
    BRIDGE_POLARIS_APPLICATION_NAME: $(Build.Repository.Name)
    BRIDGE_POLARIS_PROJECT_NAME: $(Build.Repository.Name)
```

```
    ### Accepts Multiple Values
    BRIDGE_POLARIS_ASSESSMENT_TYPES: 'SCA,SAST'
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # INCLUDE_DIAGNOSTICS: 'true'
```

Table 7. List of mandatory and optional parameters for Polaris below:

Input Parameter	Description	Mandatory / Optional
`BRIDGE_POLARIS_SERVERURL`	Polaris URL	Mandatory
`BRIDGE_POLARIS_ACCESSTOKEN`	Polaris access token	Mandatory
`BRIDGE_POLARIS_APPLICATION_NAME`	Polaris Application name	Mandatory
`BRIDGE_POLARIS_PROJECT_NAME`	Polaris Project name	Mandatory
`BRIDGE_POLARIS_ASSESSMENT_TYPES`	Polaris assessment types. Example: SCA,SAST	Mandatory

Descriptions of these arguments are shown in the Complete List of Synopsys Bridge Arguments *(on page 19)*.

Using Azure DevOps Extension with Black Duck

Synopsys Security Scan supports both self-hosted (e.g. on-prem) and Synopsys-hosted Black Duck Hub instances.

In the default Black Duck Hub permission model, projects and project versions are created on the fly and as needed. Ensure that permissions needed to create projects and project versions are granted on Black Duck Hub.

Configure sensitive data like usernames, passwords and URLs using pipeline variables. Here is an example `azure-pipelines.yml` that you can use with Black Duck:

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

variables:
```

```
  - group: blackduck


steps:
- task: SynopsysSecurityScan@1.0.0
  displayName: 'Black Duck Full Scan'
  condition: not(eq(variables['Build.Reason'], 'PullRequest'))
  ### Use below configuration to set specific detect environment variables
  env:
    DETECT_PROJECT_NAME: $(Build.Repository.Name)
  inputs:
    BRIDGE_BLACKDUCK_URL: $(BLACKDUCK_URL)
    BRIDGE_BLACKDUCK_TOKEN: $(BLACKDUCK_TOKEN)
    BRIDGE_BLACKDUCK_SCAN_FULL: true
    ### Accepts Multiple Values
    BRIDGE_BLACKDUCK_SCAN_FAILURE_SEVERITIES: 'BLOCKER,CRITICAL'
    ### Uncomment below configuration to enable automatic fix pull request creation if vulnerabilities are reported
    # BRIDGE_BLACKDUCK_AUTOMATION_FIXPR: true
    # AZURE_TOKEN: $(System.AccessToken) # Mandatory when BRIDGE_BLACKDUCK_AUTOMATION_FIXPR is set to 'true'
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # INCLUDE_DIAGNOSTICS: true


- task: SynopsysSecurityScan@1.0.0
  displayName: 'Black Duck PR Scan'
  condition: eq(variables['Build.Reason'], 'PullRequest')
  ### Use below configuration to set specific detect environment variables
  env:
    DETECT_PROJECT_NAME: $(Build.Repository.Name)
  inputs:
    BRIDGE_BLACKDUCK_URL: $(BLACKDUCK_URL)
    BRIDGE_BLACKDUCK_TOKEN: $(BLACKDUCK_API_TOKEN)
    BRIDGE_BLACKDUCK_SCAN_FULL: false
    ### Below configuration is used to enable automatic pull request comment based on Black Duck scan result
    BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT: true
    AZURE_TOKEN: $(System.AccessToken) # Mandatory when BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT is set to 'true'
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # INCLUDE_DIAGNOSTICS: true
```

Table 8. List of mandatory and optional parameters for Black Duck below:

Input Parameter	Description	Mandatory / Optional
`BRIDGE_BLACKDUCK_URL`	Black Duck URL	Mandatory
`BRIDGE_BLACKDUCK_TOKEN`	Black Duck API token	Mandatory
`BRIDGE_BLACKDUCK_INSTALL_DIRECTORY`	Installation directory for Black Duck	Optional
`BRIDGE_BLACKDUCK_SCAN_FULL`	Specifies whether full scan is required or not. Full "intelligent" scan is to be used for push events and rapid scan for pull request events. Supported values: `true` or `false`	Optional
`BRIDGE_BLACKDUCK_SCAN_FAILURE_SEVERITIES`	Black Duck scan failure severities. Supported values: `ALL, NONE, BLOCKER, CRITICAL, MAJOR, MINOR, OK, TRIVIAL, UNSPECIFIED`	Optional
`BRIDGE_BLACKDUCK_AUTOMATION_PRCOMMENT`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Black Duck PR Comment. **Note** - Feature is supported only through yaml configuration	Optional

Table 8. List of mandatory and optional parameters for Black Duck below: (continued)

Input Parameter	Description	Mandatory / Optional
`BRIDGE_BRIDGE_-BLACKDUCK_AU-TOMATION_FIXPR`	Option to enable automatic creation for fix pull requests for vulnerable direct dependencies. Default: false **Note - Black Duck automation fix pull request is currently supported for npm projects only.** **Note** - Feature is supported only through yaml configuration	Optional
`AZURE_TOKEN`	Azure Access Token Example: `AZURE_TOKEN: $(System.Access-Token)` or `AZURE_TOKEN: $(PAT_TOKEN)`	Mandatory if `BRIDGE_BLACKDUCK_AU-TOMATION_PRCOMMENT`or`BRIDGE_BRIDGE_BLACK-DUCK_AUTOMATION_FIXPR` is set true.

Note:
Detect specific options can be passed to Synopsys Bridge thru Detect environment variables.

See the Complete List of Synopsys Bridge Arguments *(on page 19)* for details of Black Duck arguments.

Using Azure DevOps Extension with Coverity Connect with Thin Client

Currently, Synopsys Security Scan only supports the Coverity Connect with thin client deployment model.

Before running Coverity Connect using the Synopsys Security Scan for Azure DevOps Extension, ensure the appropriate project and stream are set in your Coverity Connect server environment. Configure sensitive data like usernames, passwords and URLs using pipeline variables.

Here is an example `azure-pipelines.yml` that you can use to integration Coverity into your Azure pipeline:

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

variables:
  - group: coverity

steps:
- task: SynopsysSecurityScan@1.0.0
  displayName: 'Coverity Full Scan'
  condition: not(eq(variables['Build.Reason'], 'PullRequest'))
  inputs:
    BRIDGE_COVERITY_CONNECT_URL: $(COVERITY_URL)
    BRIDGE_COVERITY_CONNECT_USER_NAME: $(COVERITY_USER)
    BRIDGE_COVERITY_CONNECT_USER_PASSWORD: $(COVERITY_PASSPHRASE)
    BRIDGE_COVERITY_CONNECT_PROJECT_NAME: $(Build.Repository.Name)
    BRIDGE_COVERITY_CONNECT_STREAM_NAME: $(Build.Repository.Name)-$(Build.SourceBranchName)
    BRIDGE_COVERITY_CONNECT_POLICY_VIEW: 'Outstanding Issues'
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # include_diagnostics: true

- task: SynopsysSecurityScan@1.0.0
  displayName: 'Coverity PR Scan'
  condition: eq(variables['Build.Reason'], 'PullRequest')
  inputs:
    BRIDGE_COVERITY_CONNECT_URL: $(COVERITY_URL)
    BRIDGE_COVERITY_CONNECT_USER_NAME: $(COVERITY_USER)
    BRIDGE_COVERITY_CONNECT_USER_PASSWORD: $(COVERITY_PASSPHRASE)
    BRIDGE_COVERITY_CONNECT_PROJECT_NAME: $(Build.Repository.Name)
    BRIDGE_COVERITY_CONNECT_STREAM_NAME: $(Build.Repository.Name)-$(Build.targetBranchName)
    ### Below configuration is used to enable feedback from Coverity security testing as pull request comment
    BRIDGE_COVERITY_AUTOMATION_PRCOMMENT: true
    AZURE_TOKEN: $(System.AccessToken) # Mandatory when BRIDGE_COVERITY_AUTOMATION_PRCOMMENT is set to 'true'
    ### Uncomment below configuration if Synopsys Bridge diagnostic files needs to be uploaded
    # include_diagnostics: true
```

Table 9. List of mandatory and optional parameters for Coverity below:

Input Parameter	Description	Mandatory / Optional
`BRIDGE_COVERITY_CONNECT_URL`	Coverity URL	Mandatory
`BRIDGE_COVERITY_CONNECT_USER_NAME`	Coverity Username	Mandatory
`BRIDGE_COVERITY_CONNECT_USER_PASSWORD`	Coverity Password	Mandatory
`BRIDGE_COVERITY_CONNECT_PROJECT_NAME`	Coverity Project Name	Mandatory
`BRIDGE_COVERITY_CONNECT_STREAM_NAME`	Coverity Stream name	Mandatory
`BRIDGE_COVERITY_INSTALL_DIRECTORY`	Installation directory of Coverity	Optional
`BRIDGE_COVERITY_CONNECT_POLICY_VIEW`	ID or name of policy view to be used to enforce the “break the build” policy. If issues are found in the specified this view, build will be failed. Example: `coverity_policy_view: '100001'` or `coverity_policy_view: 'Outstanding Issues'`	Optional
`BRIDGE_COVERITY_AUTOMATION_PRCOMMENT`	Option to enable automatic creation pull request comments for new issues found in the pull request. Merge Request must be created first from feature branch to main branch to run Coverity PR Comment.	Optional

Table 9. List of mandatory and optional parameters for Coverity below: (continued)

Input Parameter	Description	Mandatory / Optional
	Default: false **Note** - Feature is supported only through yaml configuration	
`AZURE_TOKEN`	Azure Access Token Example: `AZURE_TOKEN: $(System.AccessToken)` or `AZURE_TOKEN: $(PAT_TOKEN)`	Mandatory if `BRIDGE_COVERITY_AUTOMATION_PRCOMMENT` is set true.

See the Complete List of Synopsys Bridge Arguments *(on page 19)* for details of Coverity Connect arguments.

Additional Azure Configuration

The following optional parameters can be used for Polaris, Black Duck or Coverity Connect.

- `BRIDGE_DOWNLOAD_URL`: Use this to specify the URL to Synopsys Bridge zip file to be downloaded and used .
- `BRIDGE_DOWNLOAD_VERSION`: Use this to specify the Synopsys Bridge version to use. If provided, the specified version of Synopsys Bridge will be automatically downloaded and used. If not, the latest version is downloaded and used .

> **Note:**
>
> If `bridge_download_url` is not provided, Synopsys GitHub Action downloads the latest version of Synopsys Bridge from the default SIG-REPO download location.

- `SYNOPSYS_BRIDGE_PATH`: Use this to specify the path to SynopsysBridge. Optional.

> **Note:**
>
> If this is not explicitly specified, then the integration defaults to `$HOME/synopsys-bridge`. If the installed version of Synopsys Bridge is not the latest, then the latest version of Synopsys Bridge is downloaded unless you specify the version to use explicitly (as documented below).

.

- `include_diagnostics`: When set to `true,` Synopsys Bridge diagnostic files are created. Azure DevOps no longer supports per-pipeline retention rules. The only way to configure retention policies for YAML and classic pipelines is through the project settings. For more details, see Set run retention policies.

Chapter 7. Glossary

Here are terms and concepts used by Synopsys Bridge and the various Synopsys programs with which it interfaces.

Term	Definition
Application	The software security tool used to scan code.
Application Security	Application security is enhancing software features to functionality to prevent security threats. These include denial of service attacks, unauthorized data access, privilege escalation attacks, etc. Application security is one of several levels of security used to protect systems.
BDSA	Black Duck Security Advisory, highly detailed open source vulnerability records that are hand-crafted by the Synopsys Cybersecurity Research Center (CyRC)
Black Duck	Software composition analysis (SCA) security scanning tool. Helps manage the security, quality, and license compliance risks of open source and third-party code in applications and containers. Bridge integrates with Black Duck.
CI/CD	Continuous Integration/Continuous Deployment, the process by which new checked-in code is automatically built, checked for security issues, and packaged for deployment.
CLI	Command Line Interface
Coverity	Static analysis scanning tool (SAST), which scans source code for security flaws and coding standards compliance. Bridge does not integrate with Coverity, but does integrate with Coverity Connect and CNC.
Coverity Connect	A web-based platform for Coverity. Bridge supports Coverity Connect.
Coverity cloud deployment	A cloud-native version of Coverity. Bridge supports Coverity cloud deployment, and every place in this manual that references "Coverity Connect" also applies to Coverity cloud deployment.

Term	Definition
CVE	Common Vulnerabilities and Exposures. A database of publicly identified, defined, and cataloged cybersecurity vulnerabilities.
EULM	End User License Management agreement
GUI	Graphic User Interface
IAST	Interactive application security testing (IAST) solutions help organizations identify and manage security risks associated with vulnerabilities discovered in running web applications by continuously analyzing all application interactions initiated by manual and/or automated tests to identify vulnerabilities in real time.
Polaris	Polaris is a cloud-native application security testing solution that provides both best-in-class SAST and SCA, making it easier to manage application security testing. Bridge integrates with Polaris.
Rapid Scan Static (Sigma)	Rapid Scan Static using the Sigma engine is a headless Static Application Security Testing (SAST) scanner.
RSQL	REST Query Language
Runner	An application that runs a pipeline job from a CI/CD platform like GitHub or GitLab.
SAST	Static Analysis Security Testing (SAST), or static analysis, is a testing methodology that analyzes source code to find security vulnerabilities. SAST scans an application before the code is compiled. Coverity is a SAST tool.
SCA	Software Composition Analysis (SCA) is an automated process identifying open source software in a codebase to evaluate security, license compliance, and code quality. Black Duck is an SCA tool.
SCM	Source Code Management. This usually refers to an online CI/CD SCM repo like GitHub, GitLab or Azure, all of which Synopsys offers integrations adaptors for.