SYNOPSYS®

Installing Black Duck using Kubernetes

Black Duck 2023.4.0

Contents

Preface

Black Duck documentation

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

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

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

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

- Synopsysctl is a cloud-native administration command-line tool for deploying Black Duck software in Kubernetes and Red Hat OpenShift.

Black Duck integration documentation is available on Confluence.

Customer support

If you have any problems with the software or the documentation, please contact Synopsys Customer Support.

You can contact Synopsys Support in several ways:

- Online: https://www.synopsys.com/software-integrity/support.html

- Phone: See the Contact Us section at the bottom of our support page to find your local phone number.

To open a support case, please log in to the Synopsys Software Integrity Community site at https://community.synopsys.com/s/contactsupport.

Another convenient resource available at all times is the online customer portal.

Synopsys Software Integrity Community

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

- Connect – Open support cases and monitor their progress, as well as, monitor issues that require Engineering or Product Management assistance

- Learn – Insights and best practices from other SIG product users to allow you to learn valuable lessons from a diverse group of industry leading companies. In addition, the Customer Hub puts all the latest product news and updates from Synopsys at your fingertips, helping you to better utilize our products and services to maximize the value of open source within your organization.

- Solve – Quickly and easily get the answers you're seeking with the access to rich content and product knowledge from SIG experts and our Knowledgebase.

- Share – Collaborate and connect with Software Integrity Group staff and other customers to crowdsource solutions and share your thoughts on product direction.

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

Training

Synopsys Software Integrity, Customer Education (SIG Edu) is a one-stop resource for all your Black Duck education needs. It provides you with 24x7 access to online training courses and how-to videos.

New videos and courses are added monthly.

At Synopsys Software Integrity, Customer Education (SIG Edu), you can:

- Learn at your own pace.

- Review courses as often as you wish.

- Take assessments to test your skills.

- Print certificates of completion to showcase your accomplishments.

Learn more at https://community.synopsys.com/s/education or for help with Black Duck, select **Black Duck Tutorials** from the Help menu ( ) in the Black Duck UI.

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as

our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1. Installing using Synopsysctl

Kubernetes is an orchestration tool used for managing cloud workloads through containers.

Installing using synopsysctl

Use synopsysctl to install Black Duck on Kubernetes or OpenShift.

Synopsysctl is a command line tool that assists in the deployment and management of Synopsys software in Kubernetes and OpenShift clusters. After synopsysctl is installed, you can leverage it to easily deploy and manage Synopsys software.

- Click here for an overview of synopsysctl.

- Click here for documentation about installing and using synopsysctl.

Note: For scalability sizing guidelines, see the Container Scalability section of the Black Duck Release Notes.Do not delete data from the Black Duck database (bds_hub) unless directed to do so by a Synopsys Technical Support representative. Be sure to follow appropriate backup procedures. Deletion of data will cause errors ranging from UI problems to complete failure of Black Duck to start. Synopsys Technical Support cannot recreate deleted data. If no backups are available, Synopsys will provide support on a best-effort basis.

2. Hardware requirements

The performance data below was gathered using Black Duck 2022.10.0 with reduced signature scan persistence (default) and Synopsys Detect 8.0.0. SPH values are calculated using a mix of signature scans, package manager detector scans and rapid scans. Average scan sizes vary from customer to customer so exact SPH throughput is highly customer specific. These metrics were gathered from Google Cloud Platform, which provides different database read/write IOPS for different configurations.

10sph	**Scans per hour:** 50 **SPH % Increase:** 400% **APIs per hour:** 2,500 **Project Versions:** 10,000	**IOPS:** • Read: 15,000 • Write: 9,000 **Black Duck Services:** • CPU: 12 core • Memory: 30 GB **PostgreSQL:** • CPU: 2 core • Memory: 8 GB	**Total:** • CPU: 14 core • Memory: 38 GB
120sph	**Scans per hour:** 120 **SPH % Increase:** 0% **APIs per hour:** 3,000 **Project Versions:** 13,000	**IOPS:** • Read: 15,000 • Write: 15,000 **Black Duck Services:** • CPU: 13 core • Memory: 46 GB **PostgreSQL:** • CPU: 4 core • Memory: 16 GB	**Total:** • CPU: 17 core • Memory: 62 GB
250sph	**Scans per hour:** 300 **SPH % Increase:** 20% **APIs per hour:** 7,500 **Project Versions:** 15,000	**IOPS:** • Read: 15,000 • Write: 15,000 **Black Duck Services:** • CPU: 17 core • Memory: 118 GB **PostgreSQL:** • CPU: 6 core • Memory: 24 GB	**Total:** • CPU: 23 core • Memory: 142 GB
500sph	**Scans per hour:** 650 **SPH % Increase:** 30% **APIs per hour:** 18,000 **Project Versions:** 18,000	**IOPS:** • Read: 15,000 • Write: 15,000 **Black Duck Services:**	**Total:** • CPU: 38 core • Memory: 250 GB

		• CPU: 28 core • Memory: 210 GB **PostgreSQL:** • CPU: 10 core • Memory: 40 GB	
1000sph	**Scans per hour:** 1,400 **SPH % Increase:** 40% **APIs per hour:** 26,000 **Project Versions:** 25,000	**IOPS:** • Read: 25,000 • Write: 25,000 **Black Duck Services:** • CPU: 47 core • Memory: 411 GB **PostgreSQL:** • CPU: 18 core • Memory: 72 GB	**Total:** • CPU: 65 core • Memory: 483 GB
1500sph	**Scans per hour:** 1,600 **SPH % Increase:** 6% **APIs per hour:** 41,000 **Project Versions:** 28,000	**IOPS:** • Read: 25,000 • Write: 25,000 **Black Duck Services:** • CPU: 60 core • Memory: 597 GB **PostgreSQL:** • CPU: 26 core • Memory: 104 GB	**Total:** • CPU: 92 core • Memory: 701 GB
2000sph	**Scans per hour:** 2,300 **SPH % Increase:** 15% **APIs per hour:** 50,000 **Project Versions:** 35,000	**IOPS:** • Read: 60,000 • Write: 25,000 **Black Duck Services:** • CPU: 66 core • Memory: 597 GB **PostgreSQL:** • CPU: 34 core • Memory: 136 GB	**Total:** • CPU: 100 core • Memory: 733 GB

This new guidance is based current Black Duck 2022.10.0 architecture. It is possible this guidance will be further refined for subsequent releases. If you have any questions or concerns, please reach out to Product Management.

Note: The amount of required disk space is dependent on the number of projects being managed, so individual requirements can vary. Consider that each project requires approximately 200 MB.

Black Duck Software recommends monitoring disk utilization on Black Duck servers to prevent disks from reaching capacity which could cause issues with Black Duck.

BDBA scaling is done by adjusting the number of binaryscanner replicas and by adding PostgreSQL resources based on the expected number of binary scans per hour that will be performed. For every 15 binary scans per hour, add the following:

• One binaryscanner replica

• One CPU for PostgreSQL

• 4GB memory to PostgreSQL

If your anticipated scan rate is not a multiple of 15, round up. For example, 24 binary scans per hour would require the following:

• Two binaryscanner replicas,

• Two additional CPUs for PostgreSQL, and

• 8GB additional memory for PostgreSQL.

This guidance is valid when binary scans are 20% or less of the total scan volume (by count of scans).

Binary scanning

If you are licensed for binary scanning, the uploadcache container/pod memory may need to be increased because this is where the binary scanner extracts and processes the binary. By default, the memory is set to 512MB which is not adequate for large scanning. When scanning large binaries, it is recommended to increase the memory to at least 4 GB for the uploadcache container/pod. To do so, find your override yaml and update the memory limit to 4096MB.

For Swarm installations:

```
uploadcache:
  deploy:
    resources:
      limits:
        cpus: ".200"
        memory: "4096M"
      reservations:
        cpus: ".100"
        memory: "4096M"
    replicas: 1
```

For Kubernetes installations:

```
uploadcache:
  replicas: 1
  resources:
    limits:
      cpu: "200m"
      memory: "4096Mi"
    requests:
      cpu: "100m"
      memory: "4096Mi"
```

📋 **Note:** Installing Black Duck Alert requires 1 GB of additional memory.

3. PostgreSQL versions

Black Duck 2022.10.0 supports new PostgreSQL features and functionality to improve the performance and reliability of the Black Duck service. As of Black Duck 2022.10.0, PostgreSQL container 13 is the currently supported version of PostgreSQL for the internal PostgreSQL container.

Customers upgrading from older versions of Black Duck (prior to 2022.10.0), will require a migration to PostgreSQL 13. The Black Duck 2022.10.0 update migrates the internal Black Duck PostgreSQL database container to version 13 of PostgreSQL. If you use the database container and deploy on OpenShift, you need to run a one-time migration job as documented in the Black Duck release notes and installation guide.

Note: For PostgreSQL sizing guidelines, see Black Duck Hardware Scaling Guidelines.

If you choose to run your own external PostgreSQL instance, Synopsys recommends PostgreSQL 14 for new installs. Due to an index corruption bug in PostgreSQL 14.0 through 14.3, the minimum supported version of PostgreSQL 14 is 14.4.

Note: Black Duck 2023.4.0 adds preliminary support for using PostgreSQL 15 as an external database. This support is for *testing only*; production use IS NOT SUPPORTED.

CAUTION: Do not run antivirus scans on the PostgreSQL data directory. Antivirus software opens lots of files, puts locks on files, etc. Those things interfere with PostgreSQL operations. Specific errors vary by product but usually involve the inability of PostgreSQL to access its data files. One example is that PostgreSQL fails with "too many open files in the system."

General Migration Process

The guidance here applies to upgrading from any PG 9.6 based Hub (releases prior to 2022.2.0) to 2022.10.0 or later.

1. The migration is performed by the blackduck-postgres-upgrader container.

2. If you are upgrading from a PostgreSQL 9.6-based Version of Black Duck:

   • The folder layout of the PostgreSQL data volume is rearranged to make future PostgreSQL version upgrades simpler.

   • The UID of the owner of the data volume is changed. The new default UID is 1001, but see the deployment-specific instructions.

3. The pg_upgrade script is run to migrate the database to PostgreSQL 13.

4. Plain ANALYZE is run on the PostgreSQL 13 database to initialize query planner statistics.

5. blackduck-postgres-upgrader exits.

4. Installing Black Duck using Helm

A Helm chart describes a Kubernetes set of resources that are required for Helm to deploy Black Duck. Black Duck supports Helm3 and the minimum version of Kubernetes is 1.13.

Helm charts are available here: https://sig-repo.synopsys.com/artifactory/sig-cloudnative

Click here for instructions about installing Black Duck using Helm. The Helm chart bootstraps a Black Duck deployment on a Kubernetes cluster using Helm package manager.

Migrating on Kubernetes with Helm

If you are upgrading from a PostgreSQL 9.6-based version of Black Duck, this migration replaces the use of a CentOS PostgreSQL container with a Synopsys-provided container. Also, the synopsys-init container is replaced with the blackduck-postgres-waiter container.

On plain Kubernetes, the container of the upgrade job will run as root unless overridden. However, the only requirement is that the job runs as the same UID as the owner of the PostgreSQL data volume (which is UID=26 by default).

On OpenShift, the upgrade job assumes that it will run with the same UID as the owner of the PostgreSQL data volume.

5. Artifactory Integration

Overview

The Artifactory Integration is a new mechanism to protect the Software Supply Chain. Since Artifactory is typically one of the last links of that chain, scanning each and every artifact within a configured set of Artifactory Repositories allows customers to have control of their individual supply chain. This initial version of the Artifactory Integration automatically blocks downloads from scanned Artifactory Repositories that have a Black Duck Policy Violation. Only policies that are defined as Rapid or Both will be applied to the Artifactory Integration.

📝 **Note:** Usage of the Artifactory Integration disables the Scanner and Inspector functions of the Artifactory plugin.

Deployment types

Artifactory Integration currently supports two types of deployment:

- **Full On-premise** : A fully on-premise deployment is defined as running both the Black Duck instance and the scanning service / artifactory integration in the user's on-premise environment.
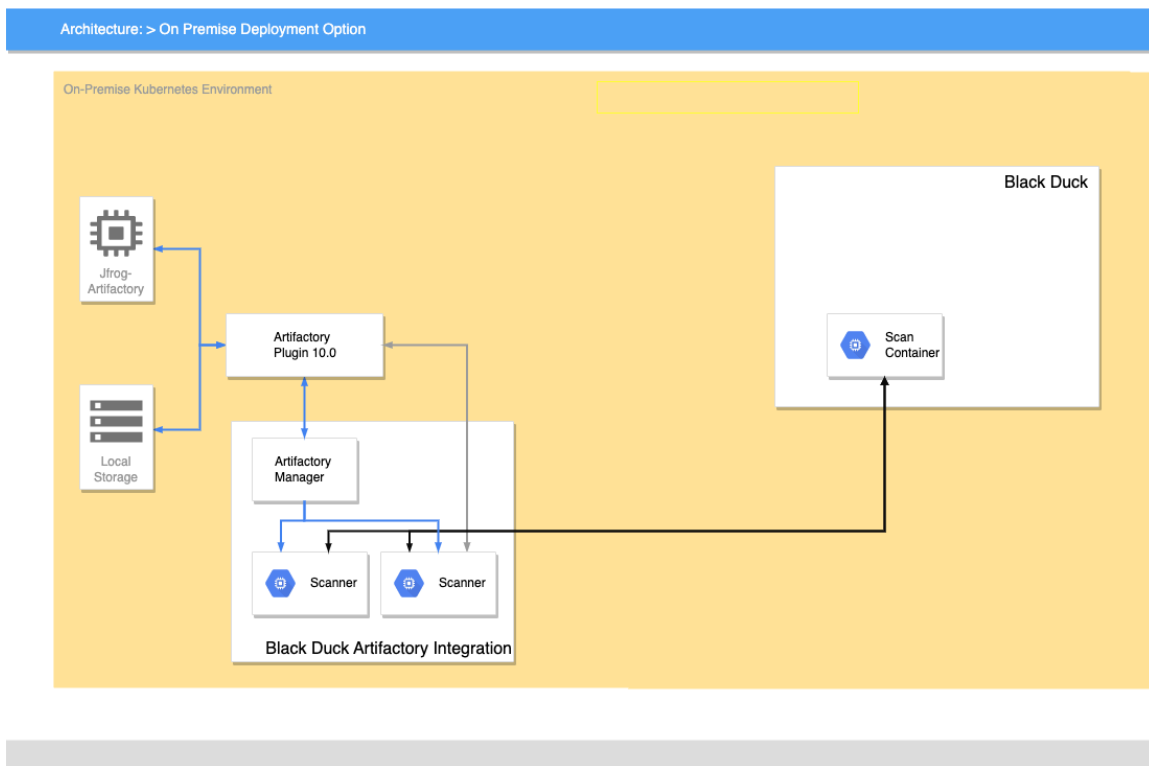
- **Hybrid**: A hybrid deployment is defined as running a Black Duck instance in our Hosted environment, but running the scanning service / artifactory integration in the user's on-premise environment.

Artifactory Integration prerequisites

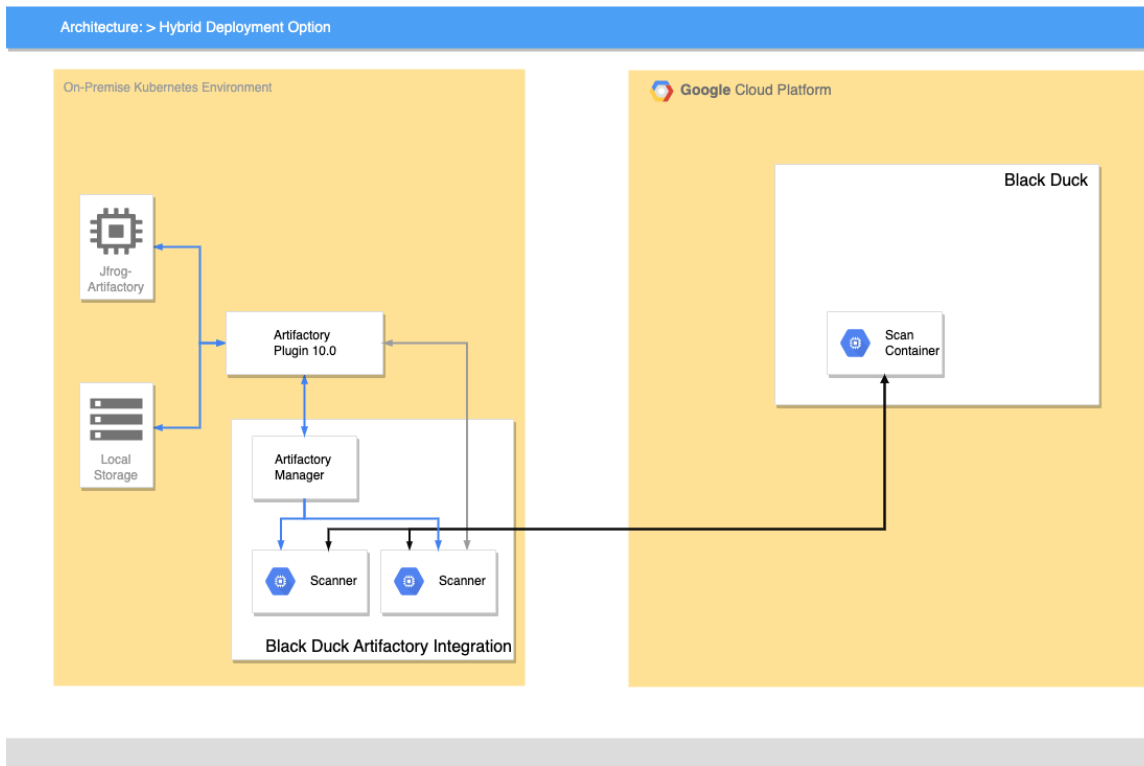| Category	Requirements

Helm and Kubernetes	• Kubernetes 1.9+
	• Helm3
	
	    • Add the Synopsys repository to Helm repository:
	
	``
	$ helm repo add synopsys https://sig-repo.synopsys.com/
	artifactory/sig-cloudnative
	``

Cluster Sizing	The recommended configuration is based upon the average size of an artifact being under 300 MB and the largest size is around 2 GB. If your particulars are significantly different, then those figures should be revisited. Likewise, the recommendation of 5 Scanners will eventually scan the entire set of configured repositories. If this size is greater than 5 TB and the speed of scanning this backlog is a concern, then more than 5 Scanners can be configured. With Scanner replicas set to 5 (Recommended): **Table 1: Cores**

	Cores Required	Total Cores
1 Manager	2 per Manager	2

Category	Requirements		
		Cores Required	**Total Cores**
	1 Message Handler	1 per Message Handler	1
	5 Scanners and BDBA workers	1 per Scanner & BDBA worker	5
			8

Table 2: Memory

		Memory Required	Total Memory
	1 Manager	4 GB per Manager	4 GB
	1 Message Handler	4 GB per Message Handler	4 GB
	5 Scanners	5 GB per Scanner	25 GB
	5 BDBA workers	5 GB per BDBA worker	25 GB
			58 GB

Storage:
Please ensure the system has enough physical disk space available for the scanner to be able to download artifacts for scanning. We recommend at least 100 GB of disk space (with scanner replicas set to 5) .

Storage Class	Artifactory Integration deployment requires a fully provisioned storage class that supports persistent volumes.
Cluster Setup	You must be able to connect to your Artifactory server from your cluster in order for the service to be able to download artifacts and annotate properties on scanned artifacts.
Persistent Volumes	Artifactory Integration deployment requires enough physical disk space available for the scanner to be able to download artifacts for scanning. We recommend at least 100 GB of disk space (with scanner replicas set to 5).
Other Requirements	**Artifactory Plugin** • BlackDuck Artifactory plugin installed in the target JFrog Artifactory Pro Server. • Installation of the plugin still follows the current instructions. Please note the configuration items of the plugin specific to the ScanAsAService module.

Order of installation

The following provides a summary of ordered steps in which the Artifactory Integration is installed:

1. Obtain an Access Token from your Black Duck instance and store in a safe location.

2. Obtain an Access Token from your Artifactory instance and store in a safe location.

3. Prepare your Kubernetes environment:

   a. Pull the latest Artifactory Integration deployment chart from sig-cloudnative.

   b. Create a namespace in Kubernetes for your deployment.

   c. Create secrets in the newly created namespace for the Black Duck and Artifactory Access Tokens.

4. Edit `values.yaml` with Artifactory Integration parameters.

5. Install Artifactory Integration into the namespace.

6. Prepare the Artifactory Plugin for installation:

    a. Download the plugin from GitHub.

    b. Decompress the downloaded file.

    c. Move the plugin file to the appropriate directories in your Artifactory installation.

7. Edit the Artifactory Plugin `blackDuckPlugin.properties` file as needed.

8. Restart your Artifactory server.

Installing Artifactory Integration

Artifactory Integration deployment via Helm

1. Pull down the latest deployment chart from sig-cloudnative repository. This downloads the deployment archive (tar.gz) that should be unpacked for further deployment steps.

    To download the latest chart:

    ```
    $ helm repo update
    $ helm pull synopsys/sca-as-a-service
    ```

    To unpack and access the chart:

    ```
    $ tar xvf sca-as-a-service-x.x.x.tgz
    $ cd sca-as-a-service
    ```

2. Create the namespace if the namespace has not been created already.

    ```
    $ BD_NAME="bd"
    $ kubectl create ns ${BD_NAME}
    ```

3. Create secrets for access tokens for both Artifactory and Black Duck instances.

    ```
    $ BD_NAME="bd"
    $ kubectl create secret generic ${BD_NAME}-scaaas-secret-store -n ${BD_NAME} --from-
    literal=scaaas-artifactory-token=<artifactory_token> --from-literal=scaaas-blackduck-
    token=<blackduck_token>
    ```

    Artifactory Access Token Scope: Can be "User", instead of "Admin" as long as the minimum permissions (listed below) are satisfied:

Permissions	Repositories
• READ • ANNOTATE	Any local or remote cache repositories that need to be scanned by Artifactory Integration.
• Deploy/Cache	Only "Local" repository where scan reports would be uploaded to.

4. Configure your Artifactory Integration deployment.

    a. Prior to installing Artifactory Integration helm chart, update the `values.yaml` file.

    b. Please set the proper values for the following environment variables (Refer to Artifactory Integration Application Configuration section below for more details).

```
environs:
    BLACKDUCK_SCAAAS_ARTIFACTORY_ANNOTATE_VIOLATING_POLICY_RULES: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_HOST: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_IGNORE_SSL: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_REPOSITORIES: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_DOCKER_REPOSITORIES: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_SCHEME: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_PORT: ""
    BLACKDUCK_SCAAAS_BLACKDUCK_HOST: ""
    BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME: ""
    BLACKDUCK_SCAAAS_BLACKDUCK_TRUST_CERTS:
    BLACKDUCK_SCAAAS_STRUCTURED_LOGGING: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_ENABLED: ""
    BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_REPOSITORY: ""
```

> 📝 **Note:** If you have any unset environment variables from the list above, it is recommended to either comment it out by adding a hashtag (#) in front it or simply deleting it.

    c. Ensure the "secrets" section is updated in order to access the secrets previously created:

```
secrets:
  artifactory:
    token:
      name: bd-scaaas-secret-store
      key: scaaas-artifactory-token
    basicAuth:
      user: {}
      password: {}
    blackduck:
      token:
        name: bd-scaaas-secret-store
        key: scaaas-blackduck-token
```

5. Deploy Artifactory Integration using helm chart.

```
$ BD_NAME="bd" && SCAAAS_NAME="scaaas"
$ helm install ${SCAAAS_NAME} sca-as-a-sevice/ --namespace ${BD_NAME}
```

Installing the Artifactory plugin

The following steps describe an overview of the process for installing and configuring the Black Duck Artifactory plugin.

1. When you've downloaded and decompressed the `blackduck-artifactory-<version>.zip` file, you're ready to configure the plugin properties file.

2. Get a Black Duck API token to use for credentials in the `blackDuckPlugin.properties` file.

3. Configure the Black Duck credentials using the `blackDuckPlugin.properties` file, which is in the `/plugins/lib` folder.

    Learn about Configuring the Plugin

4. Copy the `plugins` and the `lib` folder into `${JFrog_ARTIFACTORY_HOME}/etc/plugins/`

    which looks like the following when copied:

    `${ARTIFACTORY_HOME}/etc/plugins/lib/blackDuckPlugin.properties`

5. Change the user for the following folders:

-
  ```
  chown -R 1030:1030 <path-to-blackDuckPlugin.groovy>
  ```

-
  ```
  chown -R 1030:1030 <path-to-plugin-libs-directory>
  ```

6. Restart your Artifactory server.

Test the connection

When you install and configure the Black Duck plugin, Synopsys recommends that you test the connection and make sure the plugin works properly. Use the following curl command to test the connection.

```
curl -X GET -u USERNAME:PASSWORD "http://ARTIFACTORY_SERVER/artifactory/api/plugins/execute/
blackDuckTestConfig"
```

Artifactory installed with Docker

Perform a Docker *cp* command to move the plugin files and the *lib* folder from the extracted location to `${ARTIFACTORY_HOME}/etc/plugins/`.

Configuring the Artifactory plugin

For Black Duck Artifactory plugin versions 6.0.0 and later, all plugins and their configuration properties are incorporated into the `blackDuckPlugin.properties` file.

You must modify the `blackDuckPlugin.properties` file before the plugin can function, which you configure by manually editing the property file using any text editor. You can edit any property, including the default values. Because of this, a full reinstallation is highly recommended if you are using a plugin version lower than 6.0.0.

📝 **Note:** In Black Duck Artifactory plugin versions 6.0.0 and later, all properties referring to the Hub are removed and no longer supported.

The following is an overview of important settings the `blackDuckPlugin.properties` file.

Black Duck connection credentials

You require a connection to Black Duck that you configure in the properties file.

At a minimum, you must add the token `blackduck.api.token=<BD API token>` and the Black Duck URL under BlackDuck credentials in the property file.

```
# BlackDuck credentials
blackduck.url=
blackduck.username=
blackduck.password=
blackduck.api.token=
```

If you are using an access token and are not using a Proxy for Black Duck, this is all the information you need for the Credentials section in the properties file.

Scan-as-a-Service settings

The most important scanner setting is the repository list. You define the names of repositories Scan-as-a-Service module acknowledges. Without adjusting these settings, you will not be able to block items that are in violation of your defined Policies.

```
blackduck.artifactory.scaaas.blocking.repos=ext-release-local,libs-release
blackduck.artifactory.scaaas.blocking.docker.repos=ext-docker-repo
```

📝 **Note:** Virtual repositories are not supported as they don't contain components. The local repositories the virtual repository references should be configured to be scanned.

Another important setting is the cutoff date. If you have older applications you don't need to scan, this date defines the cutoff, where older applications than the date are ignored. Artifacts having a `lastUpdated` time prior to this value will not be subject to the blocking strategy regardless of the blocking strategy value.

`blackduck.artifactory.scan.cutoff.date=2019-05-04T00:00:00.000`

Properties file

You must modify the following properties file before the plugin can function.

```
# suppress inspection "UnusedProperty" for whole file

# BlackDuck credentials
blackduck.url=
blackduck.username=
blackduck.password=
blackduck.api.token=
blackduck.timeout=120
blackduck.proxy.host=
blackduck.proxy.port=
blackduck.proxy.username=
blackduck.proxy.password=
blackduck.trust.cert=false

# General
# The date time pattern used by the artifactory to display the scan/inspection timestamp.
# blackduck.artifactory.scan.cutoff.date must comply to this pattern
blackduck.date.time.pattern=yyyy-MM-dd'T'HH:mm:ss.SSS
blackduck.date.time.zone=

# Scanner
blackduck.artifactory.scan.enabled=false
blackduck.artifactory.scan.repos=ext-release-local,libs-release
blackduck.artifactory.scan.repos.csv.path=
blackduck.artifactory.scan.name.patterns=*.jar,*.war,*.zip,*.tar.gz,*.hpi
blackduck.artifactory.scan.binaries.directory.path=
blackduck.artifactory.scan.memory=4096
blackduck.artifactory.scan.dry.run=false
blackduck.artifactory.scan.repo.path.codelocation=true
blackduck.artifactory.scan.repo.path.codelocation.include.hostname=true
blackduck.artifactory.scan.cutoff.date=2020-05-03T00:00:00.000
blackduck.artifactory.scan.cron=0 0/1 * 1/1 * ?

# If metadata.block.repos/metadata.block.repos.csv.path is left blank, all scanned repositories
 are used

blackduck.artifactory.scan.metadata.block=false
blackduck.artifactory.scan.metadata.block.repos=
blackduck.artifactory.scan.metadata.block.repos.csv.path=

# If policy.repos/policy.repos.csv.path is left blank, all scanned repositories are used

blackduck.artifactory.scan.policy.block=true
blackduck.artifactory.scan.policy.repos=
blackduck.artifactory.scan.policy.repos.csv.path=
blackduck.artifactory.scan.policy.severity.types=BLOCKER,CRITICAL,MAJOR,MINOR,TRIVIAL,UNSPECIFIED

# Inspector
blackduck.artifactory.inspect.enabled=false
blackduck.artifactory.inspect.repos=jcenter-cache
blackduck.artifactory.inspect.repos.csv.path=
blackduck.artifactory.inspect.patterns.bower=*.tar.gz,*.tgz
blackduck.artifactory.inspect.patterns.cocoapods=*.tar.gz
blackduck.artifactory.inspect.patterns.composer=*.zip
blackduck.artifactory.inspect.patterns.conda=*.tar.bz2
```

```
blackduck.artifactory.inspect.patterns.cran=*.tar.gz,*.tgz,*.zip
blackduck.artifactory.inspect.patterns.rubygems=*.gem,*.gem.rz,*.gemspec.rz
blackduck.artifactory.inspect.patterns.maven=*.jar
blackduck.artifactory.inspect.patterns.go=*.mod,*.zip
blackduck.artifactory.inspect.patterns.gradle=*.jar
blackduck.artifactory.inspect.patterns.pypi=*.whl,*.tar.gz,*.zip,*.egg
blackduck.artifactory.inspect.patterns.nuget=*.nupkg
blackduck.artifactory.inspect.patterns.npm=*.tgz
blackduck.artifactory.inspect.cron=0 0/1 * 1/1 * ?
blackduck.artifactory.inspect.reinspect.cron=0 0 0 1/1 * ? *
blackduck.artifactory.inspect.retry.count=5
blackduck.artifactory.inspect.metadata.block=false

# If metadata.block.repos/metadata.block.repos.csv.path is left blank, all inspected repositories
 are used
blackduck.artifactory.inspect.metadata.block=false
blackduck.artifactory.inspect.metadata.block.policy.repos=
blackduck.artifactory.inspect.metadata.block.repos.csv.path=

# If policy.repos/policy.repos.csv.path is left blank, all inspected repositories are used
blackduck.artifactory.inspect.policy.block=true
blackduck.artifactory.inspect.policy.repos=
blackduck.artifactory.inspect.policy.repos.csv.path=
blackduck.artifactory.inspect.policy.severity.types=BLOCKER,CRITICAL,MAJOR,MINOR,TRIVIAL,UNSPECIFIED

# Scan-as-a-Service (scaaas)
# If Scan-as-a-Service is enabled, Scanner and Inspector *will* be disabled
blackduck.artifactory.scaaas.enabled=true
blackduck.artifactory.scaaas.blocking.strategy=BLOCK_NONE
blackduck.artifactory.scaaas.blocking.repos=ext-release-local,libs-release
blackduck.artifactory.scaaas.blocking.repos.csv.path=
blackduck.artifactory.scaaas.allowed.patterns=
blackduck.artifactory.scaaas.excluded.patterns=
# Download of items prior to this date will be ALLOWED regardless of the
# value of blackduck.artifactory.scaaas.blocking.strategy
# This date MUST comply with the format in blackduck.date.time.pattern
blackduck.artifactory.scaaas.cutoff.date=
# blocking.docker.repos and blocking.docker.repos.csv.path contain the list of repositories
# that are defined in Artifactory as Docker repositories. These MUST be specified explicitly
# and DO NOT have to be specified as part of blocking.repos or blocking.repos.csv.path.
# If empty, assumes NO repositories are of type Docker.
blackduck.artifactory.scaaas.blocking.docker.repos=ext-docker-repo
blackduck.artifactory.scaaas.blocking.docker.repos.csv.path=

# Analytics
blackduck.artifactory.analytics.enabled=true
```

Artifactory Integration tasks

Upgrading Artifactory Integration

1. Before upgrading to new version, run the commands below to pull the latest version of charts from chart museum:

```
$ helm repo update
$ helm pull synopsys/sca-as-a-service
```

2. Upgrade Artifactory Integration

```
$ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --namespace ${BD_NAME}
```

Updating Artifactory Integration

Updating is a specific type of upgrade which allows for making changes, such as adding ENV variables, to the same version. When updating it is acceptable to use the `--reuse-values` flag.

```
$ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --namespace ${BD_NAME}
```

Restarting Artifactory Integration

To restart the Artifactory Integration service, following options are available:

1. Scale down pods to "0" and then scale back to desired replicas.

   To stop:

   ```
   $ kubectl scale deployment <deployment-name> --replicas=0
   ```

   To start:

   ```
   $ kubectl scale deployment <deployment-name> --replicas=1
   ```

2. Edit status in `values.yaml`.

   Change status from "Running" to "Stopped" and then run "helm upgrade":

   ```
   $ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --set status="Stopped" --
   namespace ${BD_NAME}
   ```

   Change status from "Stopped" to "Running" and then run "helm upgrade":

   ```
   helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --set status="Stopped" --
   namespace ${BD_NAME}
   ```

Removing Artifactory Integration

To uninstall/delete the deployment:
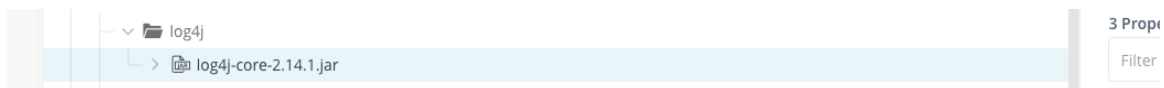
```
$  helm delete ${SCAAAS_NAME} --namespace ${BD_NAME}
```

Manual Override of Blocked Download

If the case should arise where an item in Artifactory violates a Policy you have defined in BlackDuck, but you wish to override and allow the item to be downloaded, please follow these instructions:

1. Log into the Artifactory UI and locate the in-violation item you wish to override.

2.   Select "Properties" from the right pane.

log4j-core-2.14.1.jar

General   Effective Permissions   Properties   Followers   Builds   Xray

Add: **Property**   **Property Set**

Property name	Property value

☐ Recursive ⊙

3 Properties
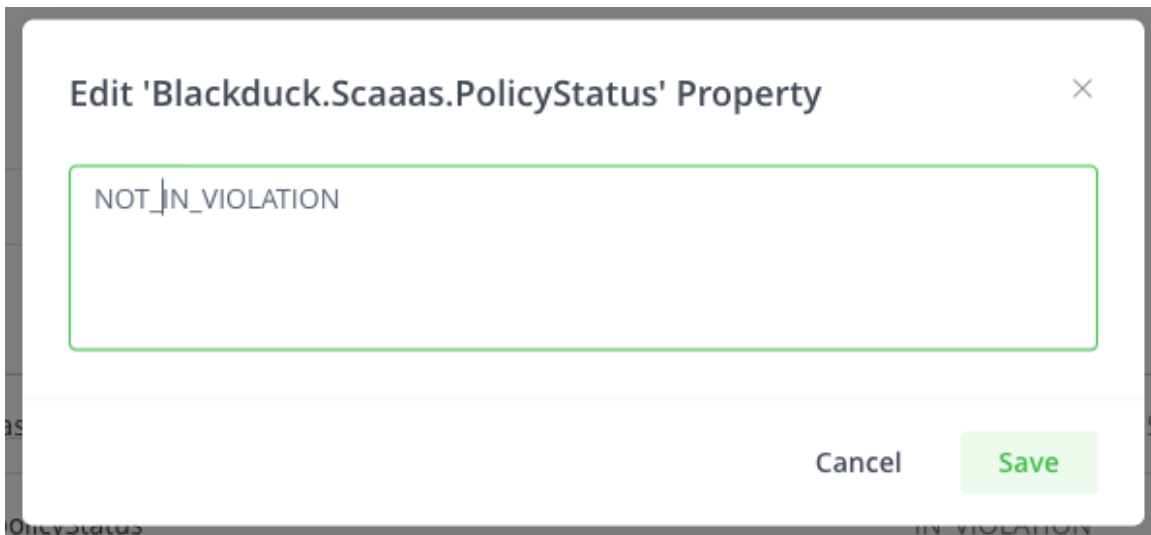
Filter

⊙	Property	Value(s)
⊙	blackduck.scaaas.lastUpdate	2023-03-28T15:19:53.755Z
⊙	blackduck.scaaas.policyStatus	IN_VIOLATION
⊙	blackduck.scaaas.scanStatus	SUCCESS

3. Select the `blackduck.scaaas.policyStatus` property and modify its value to **NOT_IN_VIOLATION** and click "Save".

Edit 'Blackduck.Scaaas.PolicyStatus' Property ✕

IN_VIOLATION

Cancel   Save

Edit 'Blackduck.Scaaas.PolicyStatus' Property ✕

NOT_IN_VIOLATION

Cancel   Save

You should now be able to download the item regardless of the Blocking Strategy set in the Artifactory Plugin.

Note: If the item is updated (eg. a new version is uploaded), it will be rescanned and the policyStatus could be set to **IN_VIOLATION** again. You will need to perform these steps again to override and allow the file to be downloaded.

Disabling binary and container scans

If your license does not permit binary and container scanning, please disable BDBA in your `values.yaml` file. This will cause the `bdbaworker` container to not be loaded or unloaded if it is already loaded. Only signature scanning will be supported.

To disable binary and container scanning, edit the "`bdbaworker`" section of your `values.yaml` file and set:

```
enabled: false
```

Save the file and follow the steps in the section above to make the changes to your deployment.

Configuring Artifactory Integration

The following table lists the configurable parameters of the Artifactory Integration chart and their default values.

Shared properties

Some properties need to be set on both Artifactory Integration and the Artifactory plugin. They should have identical values to ensure expected behavior. The following should have the same or equivalent values in both Black Duck and the Artifactory Plugin:

Exclude files

- **Black Duck Configuration**: `BLACKDUCK_SCAAAS_ARTIFACTORY_EXCLUDE_FILETYPES`
- **Artifactory Plugin Configuration**: `blackduck.artifactory.scaaas.excluded.patterns`

Include files

- **Black Duck Configuration**: `BLACKDUCK_SCAAAS_ARTIFACTORY_INCLUDE_FILETYPES`
- **Artifactory Plugin Configuration**: `blackduck.artifactory.scaaas.allowed.patterns`

Repository location

- **Black Duck Configuration**: `BLACKDUCK_SCAAAS_ARTIFACTORY_REPOSITORIES`
- **Artifactory Plugin Configuration**: `blackduck.artifactory.scaaas.blocking.repos` or `blackduck.artifactory.scaaas.blocking.repos.csv.path`

Docker Repository location

- **Black Duck Configuration**: `BLACKDUCK_SCAAAS_ARTIFACTORY_DOCKER_REPOSITORIES`
- **Artifactory Plugin Configuration**: `blackduck.artifactory.scaaas.blocking.docker.repos` or `blackduck.artifactory.scaaas.blocking.docker.repos.csv.path`

Cutoff date

- **Black Duck Configuration**: `BLACKDUCK_SCAAAS_SEARCHER_CUTOFF_DATE`
- **Artifactory Plugin Configuration**: `blackduck.artifactory.scaaas.cutoff.date`

📝 **Note:** **REQUIRED** attribute for the Artifactory Plugin to successfully initialize the Artifactory Integration module. If you do not want to set a cutoff date (basically have all items be scanned/subject to the blocking strategy), please comment out the below line in the `blackDuckPlugin.properties` file of the plugin:

`#blackduck.artifactory.scaaas.cutoff.date=`

Artifactory Integration application configuration

The following table describes the configuration items for Artifactory Integration:

Property Name	Details
`BLACKDUCK_RABBIT_SCAOP`	**Description**: Enables VHOST that is used for SCA on Polaris. **Default value**: `true` **Required**: No

Property Name	Details
	Used by: • scaaas-manager • scaaas-scanner • rabbitmq
`BLACKDUCK_RABBIT_SSL`	**Description**: Use SSL for rabbit communication. **Default value**: `false` **Required**: No **Used by**: • scaaas-manager • scaaas-scanner • rabbitmq
`BLACKDUCK_SCAAAS_ARTIFACTORY_` `ANNOTATE_VIOLATING_POLICY_RULES`	**Description**: When the property is enabled "violatingPolicyRules" annotation will be added for that artifact. **Default value**: `false` **Required**: No **Used by**: • scaaas-scanner
`BLACKDUCK_SCAAAS_ARTIFACTORY_` `EXCLUDE_FILETYPES`	**Description**: Comma-separated-value list of wildcard files to exclude when searching for items to scan. If not specified, no files in the defined repositories will be excluded. **Default value**: "*sources.jar,*sources.war" **Required**: No **Used by**: • scaaas-manager
`BLACKDUCK_SCAAAS_ARTIFACTORY_HOST`	**Description**: Name of the Artifactory host. **Default value**: N/A **Required**: Yes **Used by**: • scaaas-manager • scaaas-scanner
`BLACKDUCK_SCAAAS_ARTIFACTORY_` `IGNORE_SSL`	**Description**: Ignore SSL for Artifactory communication. **Default value**: `false` **Required**: No **Used by**: • scaaas-manager • scaaas-scanner
`BLACKDUCK_SCAAAS_ARTIFACTORY_` `INCLUDE_FILETYPES`	**Description**: Comma-separated-value list of wildcard files to include when searching for items to scan. If not specified, all files in the defined repositories will be included. **Default value**: "*.hpi,*.jar,*.tar,*.tar.gz,*.war,*.zip" **Required**: No **Used by**:

Property Name	Details
	• scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_PORT	**Description**: The port on which the proxy Artifactory host is listening. **Default value**: `-1` **Required**: No **Used by**: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ REPOSITORIES	**Description**: The repositories to look for artifacts, it is a comma separated string (aaa,bbb,ccc...). If blank, no repositories will be scanned. **Default value**: null **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ DOCKER_REPOSITORIES	**Description:** When using Artifactory as a Docker Registry, the local Docker Repositories to look for tagged images. If blank, no local Docker Repositories will be scanned. **Default value:** null **Required:** No **Used by:** scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SCAN_REPORT_ENABLED	**Description**: Enable scan report generation and storing in artifactory. **Default value**: `false` **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SCAN_REPORT_REPOSITORY	**Description**: Artifactory to use for storing scan reports. Needs to be set if `BLACKDUCK_SCAAAS_ARTIFACTORY_` `SCAN_REPORT_ENABLED` is enabled. **Default value**: blackduck-scan-results **Required**: No **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_SCHEME	**Description**: Protocol to use to connect to Artifactory. **Default value**: HTTPS **Required**: Yes **Used by**: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ SEARCHER_ADAPTIVE_QUEUE	**Description**: Adaptively increase or decrease the number of artifacts to be queued based on the current rabbitMQ queue size.

Property Name	Details
	Default value: `true` **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SEARCHER_QUEUE_SIZE	**Description**: The maximum number of artifacts to queue for scanning. **Default value**: 500 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SEARCHER_SCHEDULE_DELAY	**Description**: The interval between artifact searches in seconds. Minimum 30 seconds. **Default value**: 60 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_TOKEN	**Description**: Artifactory API token for authentication, Dev deployment only. **Default value**: N/A **Required**: Yes **Used by**: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ UPDATED_WINDOW_HOURS	**Description**: The window of how far back to search for artifacts that were previously scanned but re-uploaded after the scan date. In a very stable environment where Artifactory Integration is never down this value could be a small value like the minimum of 2 hours. **Default value**: 48 hours **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ URI_PATH	**Description**: Path to look for artifacts in Artifactory. This is appended to the end of the Artifactory hostname for all api calls and the default is 'artifactory'. **Default value**: N/A **Required**: No **Used by**: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_HOST	**Description**: Name of the Blackduck instance. **Default value**: N/A **Required**: Yes **Used by**:

Property Name	Details
	• scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_PORT	**Description**: Port on which the Black Duck instance is running. **Default value**: N/A **Required**: Yes **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME	**Description**: Protocol to use to connect to Blackduck instance. **Default value**: N/A **Required**: Yes **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_TOKEN	**Description**: Blackduck API token for scan authentication, Dev deployment only. **Default value**: N/A **Required**: No **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_ TRUST_CERTS	**Description:** Automatically trust the certificate from Black Duck instance **Default value:** false **Required:** No **Used by:** • scaaas-scanner
BLACKDUCK_SCAAAS_COMPRESS_ALL_ MQ_MESSAGES	**Description**: When enabled all rabbitMQ messages will be compressed. **Default value**: `false` **Required**: No **Used by**: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_DETECT_ BDBA_TIMEOUT	**Description**: Detect BDBA scan timeout in seconds. **Default value**: 3600 **Required**: No **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_DETECT_TIMEOUT	**Description**: Detect scan timeout in seconds. **Default value**: 600 **Required**: No **Used by**: • scaaas-scanner

Property Name	Details
BLACKDUCK_SCAAAS_DOWNLOAD_ TIMEOUT_MINUTES	**Description**: If the scanner download ends in an exception and it took longer than this time timeout, the item will be marked as FAILED. **Default value**: 30 minutes **Required**: No **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_FAILED_COUNT	**Description**: The maximum number of retries for an the artifact. **Default value**: 3 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_FAILED_ TIMEOUT_HOURS	**Description**: The number of hours to wait until allowing a scan to be declared as failed, so that it can be requeued. Allowed values: between 6 to 96 (inclusive). **Default value**: 48 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_MANAGER_SEARCHER_ QUEUE_THRESHOLD_HIGH	**Description**: The maximum threshold percentage based on the messages in the queue, if queue is full (or close), decrease queue size. **Default value**: 80 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_MANAGER_SEARCHER_ QUEUE_THRESHOLD_LOW	**Description**: The minimum threshold percentage based on the messages in the queue, if queue is empty (or close), increase queue size. **Default value**: 20 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_PROCESSING_ TIMEOUT_HOURS	**Description**: The number of hours to wait until declaring the scan in processing state to be requeued again for scanning. **Default value**: 2 **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_REPOSITORY_TYPE	**Description**: Type of the repository to scan. **Default value**: ARTIFACTORY **Required**: No **Used by**:

Property Name	Details
	• scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_REQUEST_PREFETCH	**Description**: RabbitMQ Consumer Prefetch value, to ensure how much messages consumed per consumer. **Default value**: 1 **Required**: No **Used by**: • scaaas-scanner
BLACKDUCK_SCAAAS_SEARCHER_ CUTOFF_DATE	**Description**: When configured only the artifacts created past the configured cutoff date will be considered for scanning,in the following format in the following format YYYY-MM-DDTHH:MM. **Default value**: N/A **Required**: No **Used by**: • scaaas-manager
BLACKDUCK_SCAAAS_STALL_ON_FAILURE	**Description**: By running java in the background and monitoring it we can optionally stall if java has problems launching or crashing. When enabling this option, it still have the ability to connect to the container for debugging and analysis. **Default value**: `false` **Required**: No **Used by**: • scaaas-manager • scaaas-scanner
DETECT_LATEST_RELEASE_VERSION	**Description**: Version of detect to use to scan, Artifactory Integration needs detect 8.2.0 and above. **Default value**: `<value as configured from synopsys-detect dependency>` **Required**: No **Used by**: • scaaas-scanner
RABBIT_MQ_PORT	**Description**: Port used for rabbitMQ communication. **Default value**: 5672 **Required**: Yes **Used by**: • scaaas-manager • scaaas-scanner • rabbitmq
RABBITMQ_DEFAULT_VHOST	**Description**: Virtual host to use for rabbitMQ communication. **Default value**: `blackduck` **Required**: No

Property Name	Details
	Used by: • scaaas-manager • scaaas-scanner • rabbitmq

Artifactory Plugin Configuration

The following table describes the configuration items available for Artifactory Integration in the Artifactory Plugin:

Parameter	Details
blackduck.artifactory.scaaas.enabled	Denotes whether the `ScanAsAService` Module is enabled. Setting this to `true` will force the `Scan` and `Inspect` modules to be disabled, regardless of their respective `enabled` values. **Default value**: `false`
blackduck.artifactory.scaaas. blocking.strategy	Blocking strategy employed by the plugin for download requests. The following values are supported: `BLOCK_OFF`: No blocking is applied to any download request. Actions are logged. This can be used as a "dry-run" mode or fallback mode if there are issues running the plugin otherwise. `BLOCK_NONE`: No blocking is applied to artifacts that have not yet been scanned, but blocking **will** occur only if an artifact has been scanned and have an annotation of `blackduck.scaaas.policyStatus` with value `IN_VIOLATION`. `BLOCK_ALL`: Blocking is applied to artifacts that have not yet been scanned in addition to artifacts that have been scanned and have an annotation of `blackduck.scaaas.policyStatus` with value `IN_VIOLATION`. This is the most restrictive mode. **Default value**: `BLOCK_NONE`
blackduck.artifactory.scaaas.blocking.repos	Comma-separated-valued list of repository names which will be subject to the blocking set in `blackduck.artifactory.scaaas.blocking.strategy`. This should be changed to a list of valid repositories or blocking may not properly occur. If this value is set, then `blackduck.artifactory.scaaas.blocking.repo.csv.path` should be left blank. **Default value**: ext-release-local, libs-release
blackduck.artifactory.scaaas. blocking.repos.csv.path	Path to a file containing a comma-separated-value list of repository names which will be subject to the blocking set in `blackduck.artifactory.scaaas.blocking.strategy`. If this value is set, then `blackduck.artifactory.scaaas.blocking.repo` should be left blank.
blackduck.artifactory.scaaas. blocking.docker.repos	Comma-separated-valued list of Docker repository names which will be subject to the blocking set in blackduck.artifactory.scaaas.blocking.strategy. This should be changed to a list of value Docker repositories or

Parameter	Details
	blocking may not properly occur. If thus value is set, then blackduck.artifactory.scaaas.blocking.docker.repos.csv.path should be left blank. **Default value:** ext-docker-repo
blackduck.artifactory.scaaas. blocking.docker.repos.csv.path	Path to a file containing a comma-separated-value list of Docker repository name which will be subject to the blocking set in blackduck.artifactory.scaaas.blocking.strategy. If this value is set, then blackduck.artifactory.scaaas.blocking.docker.repos should be left blank.
blackduck.artifactory.scaaas. allowed.patterns	Wildcard filter of file patterns which are subject to the blocking strategy provided. An empty value indicates **all** files are to be included. If provided, this is a comma-separated-value list.
blackduck.artifactory.scaaas. excluded.patterns	Wildcard filter of file patterns which will exclude an artifact from being subject to the blocking strategy provided. An empty value indicates **no** files are excluded. If provided, this is a comma-separated-value list.
blackduck.artifactory.scaaas.cutoff.date	Artifacts having a `lastUpdated` time *prior* to this value will **not** be subject to the blocking strategy regardless of the blocking strategy value.

6. Administrative Tasks

Configuring secrets encryption in Kubernetes

Black Duck supports encryption at rest of critical data within the system. This encryption is based upon a secret provisioned to the Black Duck installation by the orchestration environment (Docker Swarm or Kubernetes). The process to create and manage this secret, create a backup secret, and rotate the secret based upon your own organization's security policies is outlined below.

The critical data being encrypted are the following:

- SCM Integration OAuth tokens
- SCM Integration provider OAuth application client secrets
- LDAP credentials
- SAML private signing certificates

Note: Once secrets encryption is enabled, it can <u>never</u> be disabled.

What is an encryption secret?

An encryption secret is a random sequence used to generate an internal cryptographic key in order to unlock resources within the system. The encryption of secrets in Black Duck is controlled by 3 symmetric keys, the root, backup and previous keys. These three keys are generated by seeds passed into Black Duck as Kubernetes and Docker Swarm secrets. The three secrets are named:

- `crypto-root-seed`
- `crypto-backup-seed`
- `crypto-prev-seed`

In normal conditions, all three seeds will not be in active use. Unless a rotation action is in progress, the only seed active will be the root seed.

Securing the root seed

It is important to protect the root seed. A user possessing your root seed along with a copy of the system data could unlock and read the protected contents of the system. Some Docker Swarm or Kubernetes systems do not encrypt their secrets at rest by default. It is strongly advised to configure these orchestration systems to be encrypted internally so that secrets created afterwards in the system remain secure.

The root seed is necessary to recreate the system state from backup as part of a disaster recovery plan. A copy of the root seed file should be stored in a secret location separate from the orchestration system so that the combination of the seed plus the backup can recreate the system. Storing the root seed in the same location as the backup files is not advised. If one set of files is leaked or stolen – both will be, therefore, having separate locations for backup data and seed backups is recommended.

Enabling secrets encryption in Kubernetes

To enable secrets encryption in Kubernetes, you must change the value of `enableApplicationLevelEncryption` to `true` in the `values.yaml` orchestration file:

```
# if true, enables application level encryption
enableApplicationLevelEncryption: true
```

Key seed administration scripts

You can find sample administration scripts in the Black Duck GitHub public repository:

https://github.com/blackducksoftware/secrets-encryption-scripts

These scripts are not meant to be used for administering Black Duck secrets encryption, but rather to illustrate the use of the low-level Docker and Kubernetes commands documented here. There are two sets of scripts, each in its own sub-directory, corresponding to use on Kubernetes and Docker Swarm platforms. There is a one-to-one correspondence between the individual scripts, where applicable, for Kubernetes and Docker Swarm. For example, both sets of scripts contain a script called:

```
createInitialSeeds.sh
```

Generating seeds in Kubernetes

Generating seeds in OpenSSL

The content of the seeds can be generated using any mechanism that generates secure random contents at least 1024 bytes long. As soon as a seed has been created and saved in a secret, it should be removed from your file system and saved in a private, secure location.

The OpenSSL command is as follows:

```
openssl rand -hex 1024 > root_seed
```

Generating seeds in Kubernetes

There are many Kubernetes command lines that will create a secret. The command listed below allows better tracking of the secret and whether it changes or not, and ensures compatibility with being able to manipulate secrets with an online system. Secrets can be created and deleted in Kubernetes with Black Duck actively running.

```
kubectl create secret generic crypto-root-seed -n $NAMESPACE --save-config --dry-run=client --
from-file=crypto-root-seed=./root_seed -o yaml | kubectl apply -f -
```

In order to delete the prev key secret in Kubernetes:

```
kubectl delete secret crypto-prev-seed -n $NAMESPACE
```

Configuring a backup seed

Having a backup root seed is recommended to ensure the system can be recovered in a disaster recovery scenario. The backup root seed is an alternative root seed that can be put in place in order to recover a system. Consequently, it must be stored securely in the same way as a root seed.

The backup root seed has some special features in that once it is associated with the system, it remains viable even across root seed rotations. Once a backup seed is processed by the system, it should be removed from the secrets to limit its exposure to attacks and leakage. The backup root seed may have a different (less often) rotation schedule as the secret should not be "active" in the system at any point in time.

When you need or want to rotate a root seed, you first need to define the current root seed as the previous root seed. You can then generate a new root seed and put that in place.

When the system processes these seeds, the previous root key will be used to rotate resources to use the new root seed. After this processing, the previous root seed should be removed from the secrets to complete the rotation and clean up the old resources.

Creating a backup root seed

Once created initially, the backup seed/key wraps the TDEK (tenant decrypt, encrypt key) low-level key. The sample script `createInitialSeeds.sh` will create both a root and a backup seed. Once Black Duck is running, it uses both keys to wrap the TDEK.

After that operation is complete and both the root and backup seeds are securely stored elsewhere, the backup seed secret should be deleted; see sample script `cleanupBackupSeed.sh`.

If the root key is lost or leaked, the backup key can be used to replace the root key; see sample script `useRootSeed.sh`.

Rotating the backup seed

Similarly to the root key, the backup seed should be rotated periodically. Unlike for the root seed, where the old root seed is stored as a previous seed secret and a new root seed secret presented to the system, the backup seed is rotated just by creating a new backup seed. See the sample script `rotateBackupSeed.sh`.

After the rotation is complete, the new backup seed should be stored securely and removed from the Black Duck host file system.

Managing secret rotation in Kubernetes

It is good practice to rotate the root seed in use on a periodic basis according to your organization's security policy. In order to do this, an additional secret is necessary to perform the rotation. To rotate the root seed, the current root seed is configured as the "previous root seed", and a newly generated root seed is generated and configured as the root seed. Once the system processes this configuration (specifics below), the secrets will have been rotated.

At that point in time both the old and the new seeds are able to unlock the system contents. By default, the new root seed will be used, allowing you to test and make sure the system is working as intended. Once everything has been verified, you complete the rotation by removing the "previous root seed".

Once the previous root seed is removed from the system it can no longer be used to unlock the contents of the system and can be discarded. The new root seed is now the proper root seed which should be backed up and secured appropriately.

The root key is used to wrap the low-level TDEKs (tenant decrypt, encrypt key) that actually encrypt and decrypt Black Duck secrets. Periodically, at times convenient for Black Duck administrators and conforming to user organization rules, the root key should be rotated.

The procedure to rotate the root key would be create a previous seed secret with the contents of current root seed. Then a new root seed should be created and stored in the root seed secret.

Secret rotation in Kubernetes

For Kubernetes the three operations can be done with the Black Duck running. The Kubernetes sample script `rotateRootSeed.sh` will extract the root seed into `prev_root`, create a new root seed and then recreate the previous and root seeds.

After the rotation completes the previous seed secret should be removed; see sample script `cleanupPreviousSeed.sh`. Again, this cleanup can be performed on a running Kubernetes Black Duck instance.

The state of the rotation can be tracked by looking at crypto diagnostics tab, in the user interface by going to Admin > System Information > crypto.

Configuring custom volumes for Blackduck Storage

The storage container may be configured to use up to three (3) volumes for the storage of file based objects. In addition, the configuration can be set up to migrate objects from one volume to another.

Why more than one volume?

By default, the storage container uses a single volume to store all objects. This volume is sized based on typical customer usage for stored objects. As each customer is different, it may become necessary to have more space available than the volume can provide. Since not all volumes are expandable, it may become necessary to add a different, larger volume and migrate the data to the new volume.

Another reason why multiple volumes may become necessary is if the volume is hosted on a remote system (NAS or SAN) and that remote system is due to be decommissioned. A second volume hosted on a new system would need to be created and the content moved to it.

Configuring multiple volumes

To configure custom storage providers in Kubernetes, create an override file containing the following:

```
storage:
  providers:
    - name: "file-1"
      enabled: true
      index: 1
      type: "file"
      preference: 20
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "100Gi"
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads"
    - name: "file-2"
      enabled: true
      index: 2
      type: "file"
      preference: 10
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "200Gi"
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads2"
    - name: "file-3"
      enabled: false
      index: 3
      type: "file"
      preference: 30
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "100Gi"
```

```
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads3"
```

In the above override file both providers 1 and 2 are enabled with provider 2 having a higher priority (lower preference number) and so all new content is directed there.

The possible settings for each provider are as follows:

Setting	Details
name	**Default:** none. **Valid values**: any. **Notes**: This is a cosmetic label to assist in administration of these providers.
enabled	**Default:** `true` for provider 1, `false` for others. **Valid values**: `true` or `false`. **Notes**: Indicates if the provider is enabled or not.
index	**Default:** none. **Valid values**: `1`, `2`, `3`. **Notes**: Indication of the provider number. The sequence in the configuration file does not matter.
type	**Default:** `file`. **Valid values**: `file`. **Notes**: `"file"` is the only supported provider type.
preference	**Default:** `index` times 10. **Valid values**: `0-999`. **Notes**: Sets the preference of the provider. Providers with the highest priority (lowest preference number) will have new content added to them. NOTE: All provider preferences must be unique, Two providers cannot share the same value.
readonly	**Default:** `false`. **Valid values**: `true` or `false`. **Notes**: Indicates a provider is read-only. The highest priority (lowest preference number) provider cannot be read-only or the system cannot function. A read only provider will not have the storage volume altered by addition of data or removal of data, however metadata in the database will be manipulated to record object deletions and other changes.
migrationMode	**Default:** `none`. **Valid values**: `none`, `drain`, `delete`, `duplicate`. **Notes**: Configures the migration mode for the provider, Details of what this mode is and how to use it are provided in the migration section of this document.
existingPersistentVolumeClaimName	**Default:** `""`. **Valid values**: any valid k8s identifier.

Setting	Details
	Notes: Allows you to specify a specific persistence volume claim name for this volume.
`pvc.size`	**Default:** `none.` **Valid values**: any valid size. **Notes**: Allows you to specify the amount of space available to the volume.
`pvc.storageClass`	**Default:** `""`. **Valid values**: any valid k8s identifier. **Notes**: Allows you to specify a specific storage class for this volume.
`pvc.existingPersistentVolumeName`	**Default:** `""`. **Valid values**: any valid k8s identifier. **Notes**: Allows you to specify a specific persistence volume name for this volume.
`mountPath`	**Default:** specific to index - see notes. **Valid values**: /opt/blackduck/hub/uploads /opt/blackduck/hub/uploads2 /opt/blackduck/hub/uploads3 **Notes**: Sets the mount point for a specific provider. A provider with index one (1) must specify the mount point /opt/blackduck/hub/uploads. A provider with index two (2) must specific the mount point /opt/blackduck/hub/uploads2. A provider with index three (3) must specify the mount point /opt/blackduck/hub/uploads3

Migrating Between Volumes

With multiple volumes configured, it is possible to migrate content from one or more provider volumes to a new provider volume. This can only be done for providers that are not the highest priority (lowest preference). To do this, configure the volumes with one of the following migration modes. Once configured, Black Duck needs to be restarted in order to initiate the migration which is performed by a job in the background until it is completed.

Migration Mode	Details
`none`	**Purpose**: To indicate no migration is in progress. **Notes**: The default migration mode.
`drain`	**Purpose**: This mode moves content from the configured provider to the highest priority (lowest preference number) provider. Once content is moved, it is removed immediate from the source provider. **Notes**: This is a straight move operation - adding it to the target provider and removing it from the source.

Migration Mode	Details
delete	**Purpose**: This mode copies content from the configured provider to the highest priority (lowest preference number) provider. Once content is copied, it is marked for deletion in the source provider. The standard deletion retention periods apply - after that period the content is removed. **Notes**: This is a move that allows for the ability for the system to be recovered from backup within the retention window so that content in the source provider remains viable. The default retention period is 6 hours.
duplicate	**Purpose**: This mode copies content from the configured provider to the highest priority (lowest preference number) provider. Once content is copied, the source is left unaltered, including the metadata. **Notes**: After the duplicate migration, you will have two volumes with all of the content and the metadata in the database. If you take the next step in the "duplicate and dump" process and unconfigure the original volume, the files will be deleted but the metadata will remain in the database - referencing an unknown volume generating a warning in the pruner jobs (a job error). To resolve the error, use the following property to enable the pruning of the orphaned metadata records: `storage.pruner.orphaned.data.pruning.enable=true`

Configuring jobrunner thread pools

In Black Duck, there are two job pools - one that runs scheduled jobs (called the periodic pool) and one that runs jobs that are initiated from some event, including API or user interactions (called the ondemand pool).

Each pool has two settings: max threads, and prefetch.

Max threads is the maximum number of jobs a jobrunner container can run at the same time. Adding together periodic and ondemand max threads should never be larger than 32 as most jobs use the database and there are at most 32 connections. It is very easy to saturate the jobrunner memory, so the default thread counts are set very low.

Prefetch is the number of jobs each jobrunner container with grab in each round trip to the database. Setting this higher is more efficient, but setting it lower will spread the load more evenly across multiple jobrunners (although even load is not a design goal of jobrunner in general).

In Kubernetes, you can override the thread counts settings using the following override file:

```
jobrunner:
    maxPeriodicThreads: 2
    maxPeriodicPrefetch: 1
    maxOndemandThreads: 4
    maxOndemandPrefetch: 2
```