Installing Black Duck using Kubernetes and OpenShift

Black Duck 2024.7.1

04-09-2024

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.
Installing Black Duck using Kubernetes	install_kubernetes.pdf	Contains information about installing and upgrading Black Duck using Kubernetes.
Installing Black Duck using OpenShift	install_openshift.pdf	Contains information about installing and upgrading Black Duck using OpenShift.
Getting Started	getting_started.pdf	Provides first-time users with information on using Black Duck.
Scanning Best Practices	scanning_best_practices.pdf	Provides best practices for scanning.
Getting Started with the SDK	getting_started_sdk.pdf	Contains overview information and a sample use case.
Report Database	report_db.pdf	Contains information on using the report database.
User Guide	user_guide.pdf	Contains information on using Black Duck's UI.

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

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

Black Duck integration documentation is available on:

- https://sig-product-docs.synopsys.com/bundle/integrations-detect/page/integrations/integrations.html
- https://sig-product-docs.synopsys.com/category/cicd_integrations

Customer support

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

You can contact Black Duck 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 Black Duck Software Integrity Community site at https://community.synopsys.com/s/contactsupport.

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

Black Duck Software Integrity Community

The Black Duck 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 Black Duck at your fingertips, helping you to better utilize our products and services to maximize the value of open source within your organization.
- Solve – Quickly and easily get the answers you're seeking with the access to rich content and product knowledge from 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

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

New videos and courses are added monthly.

At Black Duck Software Integrity Group (SIG), Customer Education, you can:

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

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

Black Duck Statement on Inclusivity and Diversity

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

Black Duck Security Commitments

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

This statement is available at: Security Commitments | Black Duck

1. Installing Black Duck using Kubernetes and OpenShift

Kubernetes and OpenShift™ are orchestration tools used for managing cloud workloads through containers.

⚠ **Warning:** As of the Black Duck 2023.7.0 release, Black Duckctl is no longer supported and there will be no more updates. Black Duck deployments are supported via Helm chart, see documentation as well as Helm chart samples, in %install dir%/kubernetes/blackduck/.

2. Hardware requirements

Black Duck hardware scaling guidelines

For scalability sizing guidelines, see Black Duck Hardware Scaling Guidelines.

Black Duck database

⚠ **DANGER:** Do not delete data from the Black Duck database (`bds_hub`) unless directed to do so by a Black Duck 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. Black Duck Technical Support cannot recreate deleted data. If no backups are available, Black Duck will provide support on a best-effort basis.

Disk space requirements

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

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).

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

3. PostgreSQL versions

Black Duck 2023.10.0 supports new PostgreSQL features and functionality to improve the performance and reliability of the Black Duck service. As of Black Duck 2023.10.0, PostgreSQL 14 is the supported version of PostgreSQL for the internal PostgreSQL container.

Starting with Black Duck 2023.10.0, PostgreSQL settings will be automatically set in deployments using the PostgreSQL container. Customers using external PostgreSQL will still need to apply the settings manually.

Customers using the PostgreSQL container and upgrading from versions of Black Duck between 2022.2.0 and 2023.7.x (inclusive), will be automatically migrated to PostgreSQL 14. Customers upgrading from older versions of Black Duck will need to upgrade to 2023.7.x before upgrading to 2024.7.0.

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

If you choose to run your own external PostgreSQL instance, Black Duck recommends the latest version PostgreSQL 16 for new installs.

Note: Black Duck 2024.4.0 added preliminary support for using PostgreSQL 16 as an external database for testing only; beginning with Black Duck 2024.7.0, PostgreSQL 16 is fully supported for production use.

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 Helm 3.5.4 and the minimum version of Kubernetes is 1.17.

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 Black Duck-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. 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 never 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
```

Configuring readiness probes

You can enable or disable the readiness probes by editing the following boolean flags in `values.yaml`:

```
enableLivenessProbe: true
enableReadinessProbe: true
enableStartupProbe: true
```

Configuring HUB_MAX_MEMORY setting

The configuration parameter `HUB_MAX_MEMORY` is automatically set for relevant containers in Kubernetes-based deployments. The value is computed as a percentage of the memory limit, with 90% being the default.

In the gen04 deployment sizings, the `maxRamPercentage` controls the percentage used; the values for this setting were chosen so that `HUB_MAX_MEMORY` has the same values as before.

Migrating on OpenShift 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 Black Duck-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.