

Installing Black Duck SCA using Kubernetes and OpenShift

Black Duck SCA 2026.7.2

Copyright ©2026 by Black Duck.

All rights reserved. All use of this documentation is subject to the license agreement between Black Duck Software, Inc. and the licensee. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the prior written permission of Black Duck Software, Inc.

Black Duck, Know Your Code, and the Black Duck logo are registered trademarks of Black Duck Software, Inc. in the United States and other jurisdictions. Black Duck Code Center, Black Duck Code Sight, Black Duck Hub, Black Duck Protex, and Black Duck Suite are trademarks of Black Duck Software, Inc. All other trademarks or registered trademarks are the sole property of their respective owners.

03-11-2026

Contents

Preface....5
Black Duck documentation....5
Customer support....5
Black Duck Community....6
Training....6
Black Duck Statement on Inclusivity and Diversity....6
Black Duck Security Commitments....7

1. Installing Black Duck using Kubernetes and OpenShift....8

2. Hardware requirements....9

3. PostgreSQL Versions Supported by Black Duck SCA....10

4. Installing Black Duck using Helm....11
Black Duck helm chart....11
Getting started....11
Configuring your Black Duck instance....12
Exposing the Black Duck UI....13
Installing the chart....14
Uninstalling the chart....15
Upgrading the chart....15
Configuration parameters....15
Common configuration....16
Authentication pod configuration....18
Binary scanner pod configuration....19
BOM engine pod configuration....20
CFSSL pod configuration....20
Documentation pod configuration....21
Integration pod configuration....21
JDBC Connection Pool Configuration....22
Job runner pod configuration....23
Logstash pod configuration....24
Match engine pod configuration....25
PostgreSQL pod configuration....25
PostgreSQL readiness init container configuration....27
PostgreSQL upgrade job configuration....27
RabbitMQ pod configuration....27
Redis pod configuration....28
Registration pod configuration....28
Scan pod configuration....29
Storage pod configuration....30
Webapp pod configuration....32
Webserver pod configuration....33

5. Administrative Tasks 34

Configuring secrets encryption in Kubernetes 34
Generating seeds in Kubernetes 35
Configuring a backup seed 35
Managing secret rotation in Kubernetes 36
Passing external database credentials via Kubernetes secret 37
Configuring custom volumes for Blackduck Storage 37
Configuring jobrunner thread pools 41
Configuring readiness probes 41
Configuring HUB_MAX_MEMORY setting 41
Migrating on OpenShift with Helm 42
Provisioning JWT public/private key pairs 42
Enabling SCM Integration 43
Configuring mTLS on an external database 43
Transitioning from Synopsysctl to helm chart deployments 45

Preface

Black Duck documentation

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

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

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

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

Black Duck integration documentation is available on:

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

Customer support

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

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

Black Duck Community

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

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

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

Training

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

New videos and courses are added monthly.

In Black Duck Education, you can:

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

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

Black Duck Statement on Inclusivity and Diversity

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

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

Black Duck Security Commitments

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

This statement is available at: Security Commitments | Black Duck

1. Installing Black Duck using Kubernetes and OpenShift

Kubernetes and OpenShift™ are orchestration tools used for managing cloud workloads through containers. Black Duck deployments are supported via Helm chart, see documentation as well as Helm chart samples, in %install dir%/kubernetes/blackduck/.

2. Hardware requirements

Supported systems

Black Duck supports the following systems for installation and operation:

- 64-bit x86
- ARM64 (AArch64)

Note: BDBA and RL-service are currently not supported on ARM systems.

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 Supported by Black Duck SCA

Black Duck SCA continuously updates its support for PostgreSQL versions to enhance the performance and reliability of its services. This document summarizes supported versions for both internal PostgreSQL containers and external PostgreSQL instances, along with migration guidance.

Supported PostgreSQL Versions

Internal PostgreSQL Container:

- As of **Black Duck SCA 2025.10.0**, PostgreSQL 16 is the supported version for the internal PostgreSQL container.
- Starting with **Black Duck SCA 2023.10.0**, PostgreSQL settings are automatically configured for deployments using the internal PostgreSQL container.

External PostgreSQL Instances:

- For new external PostgreSQL installations, Black Duck recommends using the latest stable version, **PostgreSQL 18**.

Important Notes

- **PostgreSQL Sizing:** Refer to the Black Duck SCA Hardware Scaling Guidelines for sizing recommendations.
- **Antivirus Caution:** Avoid running antivirus scans on the PostgreSQL data directory. Antivirus software may lock files and interfere with database operations, potentially causing errors such as "too many open files in the system."

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://repo.blackduck.com/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 blackduck-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.

Black Duck helm chart

This chart bootstraps Black Duck deployment on a Kubernetes cluster using the Helm package manager.

Note: This document describes a quickstart process of installing a basic deployment. For more configuration options, please refer to the Kubernetes documentation.

Prerequisites

- Kubernetes 1.16+
  - A `storageClass` configured that allows persistent volumes.

    The `reclaimPolicy` of the `storageClass` in use should be set to `Retain` to ensure data persistence. AzureFile (non-CSI variant) requires a custom storage class for RabbitMQ due to it being treated as an SMB mount where file and directory permissions are immutable once mounted into a pod.
- Helm 3
- Adding the repository to your local Helm repository:

```
$ helm repo add blackduck https://repo.blackduck.com/cloudnative
```

Getting started

Save the chart locally

To save the chart on your machine, run the following command:

```
$ helm pull blackduck/blackduck -d <DESTINATION_FOLDER> --untar
```

This will extract the charts to the specified folder (as denoted by the `-d` flag in the above command), which contains the necessary files to deploy the application.

Create a namespace

Create a namespace by running the commands below. The example uses `bd`, but you can replace it with any name of your choice.

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

Create a custom TLS secret (optional)

It's common to provide a custom web server TLS secret before installing the Black Duck Helm chart. Create the secret with the command below:

```
$ BD_NAME="bd"
$ kubectl create secret generic ${BD_NAME}-blackduck-webserver-certificate -n ${BD_NAME} --from-
file=WEBSERVER_CUSTOM_CERT_FILE=tls.crt --from-file=WEBSERVER_CUSTOM_KEY_FILE=tls.key
```

Next, update the `TLS certificate for Black Duck` block in `values.yaml`, ensuring to uncomment the `tlsCertSecretName` value (tls.crt and tls.key files are required). If the value `tlsCertSecretName` is not provided then Black Duck will generate its own certificates.

```
tlsCertSecretName: ${BD_NAME}-blackduck-webserver-certificate
```

Note: This step is not required where TLS termination is being handled upstream from the application (i.e. via an ingress resource).

Configuring your Black Duck instance

Choosing an appropriate deployment size

Black Duck provides several Black Duck pre-configured scans-per-hour yaml files to help with sizing your deployment appropriately. These have been tested by our performance lab using real-world configurations. However, they are not "one size fits all", therefore, if you plan to run large amounts BDBA scans, snippet scans or reports, please reach out to your CSM for assistance in determining a custom sizing tier.

As of 2024.4.x, GEN04 sizing files should be used.

Note: The 10sph.yaml files are not intended for production purposes and should not be deployed for anything outside of local testing.

Configuring persistent storage

Black Duck requires certain data to be persisted to disk. Therefore, an appropriate `storageClass` should be utilized within your install. If your cluster does not have a default `storageClass`, or you wish to override it, update the following parameters:

```
# it will apply to all PVC's storage class but it can be override at container level
storageClass:
```

The `reclaimPolicy` of the `storageClass` in use should be set to `Retain` to ensure data persistence. AzureFile (non-CSI variant) requires a custom storage class for RabbitMQ due to it being treated as an SMB mount where file and directory permissions are immutable once mounted into a pod.

Configuring the database

If you choose to use an external postgres instance (default configuration), you will need to configure the following parameters in values.yaml:

```
postgres.host: ""
postgres.adminUsername: ""
postgres.adminPassword: ""
postgres.userUsername: ""
postgres.userPassword: ""
```

If you choose to utilize the containerized PostgreSQL instance, set the following parameter to false:

```
postgres.isExternal: true
```

Note: It is important that the specifications of the database deployment meet the appriopriate size tier. Regardless of whatever database deployment method you choose, ensure that you regularly perform backups and periodically verify the integrity of those backups.

Exposing the Black Duck UI

The Black Duck User Interface (UI) can be accessed via several methods, described below.

NodePort

`NodePort` is the default service type set in the values.yaml. If you want to use a custom NodePort, set the following parameters in values.yaml to the desired port:

```
# Expose Black Duck's User Interface
exposeui: true
# possible values are NodePort, LoadBalancer or OpenShift (in case of routes)
exposedServiceType: NodePort
# custom port to expose the NodePort service on
exposedNodePort: "<NODE_PORT_TO_BE_USED>"
```

You can access the Black Duck UI via `https://${NODE_IP}:${NODE_PORT}`.

Load balancer

Setting the `exposedServiceType` to LoadBalancer in the values.yaml, will instruct Kubernetes to deploy an external Load Balancer service. You can use the following command to get the external IP address of the Black Duck web server:

```
$ kubectl get services ${BD_NAME}-blackduck-webserver-exposed -n ${BD_NAME}
```

If the external IP address is shown as pending, wait for a minute and enter the same command again.

You can access the Black Duck UI by https://${EXTERNAL_IP}.

Ingress

This is typically the most common method of exposing the application to users. Firstly, set `exposeui` in the values.yaml to `false` since the ingress will route to the service.

```
# Expose Black Duck's User Interface
exposeui: false
```

A typical ingress manifest would be representative of the example below. Note, the configuration of the ingress controller and TLS certificates themselves are outside of the scope of this guide.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ${BD_NAME}-blackduck-webserver-exposed
  namespace: ${BD_NAME}
spec:
  rules:
  - host: blackduck.foo.org
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ${BD_NAME}-blackduck-webserver
            port:
              number: 443
  ingressClassName: nginx
```

Once deployed, the UI will be available on port 443 on the Public IP of your ingress controller.

OpenShift

Setting the `exposedServiceType` to OpenShift in the values.yaml, will instruct OpenShift to deploy route service.

```
# Expose Black Duck's User Interface
exposeui: true
# possible values are NodePort, LoadBalancer or OpenShift (in case of routes)
exposedServiceType: OpenShift
```

You can use the following command to get the OpenShift routes:

```
$ oc get routes ${BD_NAME}-blackduck -n ${BD_NAME}
```

You can access the Black Duck UI by `https://${ROUTE_HOST}`.

Installing the chart

To install the Black Duck chart, run the following command:

```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph" && BD_INSTALL_DIR="<DESTINATION_FOLDER>/
blackduck/"
$ helm install ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/
values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml
```

Tip: List all releases using `helm list` and list all specified values using `helm get values` RELEASE_NAME.

You must not use the `--wait` flag when you install the Helm Chart. The `--wait` flags waits for all pods to become Ready before marking the **Install** as done. However the pods will not become Ready until the postgres-init job is run during the **Post-Install**. Therefore the **Install** will never finish.

Alternatively, Black Duck can be deployed using `kubectl apply` by generating a dry run manifest from Helm:

```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph" && BD_INSTALL_DIR="<DESTINATION_FOLDER>/
blackduck/"
$ helm install ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/
values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml --dry-run=client > ${BD_NAME}.yaml

# install the manifest
$ kubectl apply -f ${BD_NAME}.yaml --validate=false
```

Uninstalling the chart

To uninstall/delete the deployment:

```
$ helm uninstall ${BD_NAME} --namespace ${BD_NAME}
```

The command removes all the Kubernetes components associated with the chart and deletes the release.

If you have used `kubectl` to install from a dry-run as shown above, the following command will remove the install:

```
$ kubectl delete -f ${BD_NAME}.yaml
```

Upgrading the chart

Before upgrading to new version, please make sure to run the following command to pull the latest version of charts from chart museum:

```
$ helm repo update
$ helm pull blackduck/blackduck -d <DESTINATION_FOLDER> --untar
```

To update the deployment:

```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph"
$ helm upgrade ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/
values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml
```

Configuration parameters

The following tables list the configurable parameters of the Black Duck chart and their default values.

- Common configuration
- Authentication pod
- Binary scanner pod
- BOM engine pod
- CFSSL pod
- Integration pod
- Job runner pod
- Logstash pod
- Match engine pod
- PostgreSQL pod

- PostgreSQL readiness init container
- PostgreSQL upgrade job
- RabbitMQ pod
- Redis pod
- Registration pod
- Scan pod
- Storage pod
- Webapp pod
- Webserver pod

Note: Do not set the following parameters in the environs flag. Instead, use their respective flags.

```
Use dataRetentionInDays, enableSourceCodeUpload and maxTotalSourceSizeinMB for the following:
* DATA_RETENTION_IN_DAYS
* ENABLE_SOURCE_UPLOADS
* MAX_TOTAL_SOURCE_SIZE_MB

Use enableAlert, alertName and alertNamespace for the following:
* USE_ALERT
* HUB_ALERT_HOST
* HUB_ALERT_PORT

Use exposedNodePort and exposedServiceType for the following:
* PUBLIC_HUB_WEBSERVER_PORT

Use postgres.isExternal and postgres.ssl for the following:
* HUB_POSTGRES_ENABLE_SSL
* HUB_POSTGRES_ENABLE_SSL_CERT_AUTH

Use enableIPV6 for the following:
* IPV4_ONLY
```

Common configuration

Parameter	Description	Default
`registry`	Image repository	`docker.io/blackducksoftware`
`imageTag`	Version of Black Duck	`2024.7.1`
`imagePullSecrets`	Reference to one or more secrets to be used when pulling images	`[]`
`tlsCertSecretName`	Name of Webserver TLS Secret containing Certificates (if not provided Certificates will be generated)	
`exposeui`	Enable Black Duck Web Server User Interface (UI)	`true`
`exposedServiceType`	Expose Black Duck Web Server Service Type	`NodePort`
`enablePersistentStorage`	If true, Black Duck will have persistent storage	`true`

Parameter	Description	Default
`storageClass`	Global storage class to be used in all Persistent Volume Claim	
`enableLivenessProbe`	If true, Black Duck will have liveness probe	`true`
`enableInitContainer`	If true, Black Duck will initialize the required databases	`true`
`enableSourceCodeUpload`	If true, source code upload will be enabled by setting in the environment variable (this takes priority over environs flag values)	`false`
`dataRetentionInDays`	Source code upload's data retention in days	`180`
`maxTotalSourceSizeinMB`	Source code upload's maximum total source size in MB	`4000`
`enableBinaryScanner`	If true, binary analysis will be enabled by deploying the binary scan worker	`false`
`enableIntegration`	If true, blackduck integration will be enabled by setting in the environment variable (this takes priority over environs flag values)	`false`
`enableAlert`	If true, the Black Duck Alert service will be added to the nginx configuration with the environ `"HUB_ALERT_HOST:<blackduck_name>-alert.<blackduck_name>.svc`	`false`
`enableIPV6`	If true, IPV6 support will be enabled by setting in the environment variable (this takes priority over environs flag values)	`true`
`certAuthCACertSecretName`	Own Certificate Authority (CA) for Black Duck Certificate Authentication	`run this command "kubectl create secret generic -n <namespace> <name>-blackduck-auth-custom-ca --from-file=AUTH_CUSTOM_CA=ca.crt" and provide the secret name`
`proxyCertSecretName`	Black Duck proxy server's Certificate Authority (CA)	`run this command "kubectl create secret generic -n <namespace> <name>-blackduck-proxy-certificate --from-file=HUB_PROXY_CERT_FILE=proxy.crt" and`

Parameter	Description	Default
		provide the secret name
proxyPasswordSecretName	Black Duck proxy password secret	run this command "kubectl create secret generic -n <namespace> <name>-blackduck-proxy-password --from-file=HUB_PROXY_PASSWORD_FILE=proxy_password and provide the secret name
ldapPasswordSecretName	Black Duck LDAP password secret	run this command "kubectl create secret generic -n <namespace> <name>-blackduck-ldap-password --from-file=LDAP_TRUST_STORE_PASSWORD_FILE=ldap_pa and provide the secret name
environs	environment variables that need to be added to Black Duck configuration	map e.g. if you want to set PUBLIC_HUB_WEBSERVER_PORT, then it should be --set environs.PUBLIC_HUB_WEBSERVER_PORT=30269

Authentication pod configuration

Parameter	Description	Default
authentication.registry	Image repository to be override at container level	
authentication.resources.limits.memory	Authentication container Memory Limit	1024Mi
authentication.resources.requests.memory	Authentication container Memory request	1024Mi
authentication.maxRamPercentage	Authentication container maximum heap size	90
authentication.persistentVolumeClaimName	Pod to use existing Authentication Persistent Volume Claim (PVC)	
authentication.claimSize	Authentication Persistent Volume Claim (PVC) claim size	2Gi
authentication.storageClass	Authentication Persistent Volume Claim (PVC) storage class	

Parameter	Description	Default
`authentication.volumeName`	Point to an existing Authentication Persistent Volume (PV)	
`authentication.nodeSelector`	Authentication node labels for pod assignment	`{}`
`authentication.tolerations`	Authentication node tolerations for pod assignment	`[]`
`authentication.affinity`	Authentication node affinity for pod assignment	`{}`
`authentication.podSecurityContext`	Authentication security context at pod level	`{}`
`authentication.securityContext`	Authentication security context at container level	`{}`

Binary scanner pod configuration

Parameter	Description	Default
`binaryscanner.registry`	Image repository to be override at container level	`docker.io/sigblackduck`
`binaryscanner.imageTag`	Image tag to be override at container level	`2024.6.3`
`binaryscanner.resources.limits.cpu`	Binary Scanner container CPU Limit	`1000m`
`binaryscanner.resources.requests.cpu`	Binary Scanner container CPU request	`1000m`
`binaryscanner.resources.limits.memory`	Binary Scanner container Memory Limit	`2048Mi`
`binaryscanner.resources.requests.memory`	Binary Scanner container Memory request	`2048Mi`
`binaryscanner.nodeSelector`	Binary Scanner node labels for pod assignment	`{}`
`binaryscanner.tolerations`	Binary Scanner node tolerations for pod assignment	`[]`
`binaryscanner.affinity`	Binary Scanner node affinity for pod assignment	`{}`
`binaryscanner.podSecurityContext`	Binary Scanner security context at pod level	`{}`
`binaryscanner.securityContext`	Binary Scanner security context at container level	`{}`

BOM engine pod configuration

Parameter	Description	Default
`bomengine.registry`	Image repository to be override at container level	
`bomengine.resources.limits.memory`	BOM Engine container Memory Limit	`1024Mi`
`bomengine.resources.requests.memory`	BOM Engine container Memory request	`1024Mi`
`bomengine.maxRamPercentage`	BOM Engine container maximum heap size	`90`
`bomengine.nodeSelector`	BOM Engine node labels for pod assignment	`{}`
`bomengine.tolerations`	BOM Engine node tolerations for pod assignment	`[]`
`bomengine.affinity`	BOM Engine node affinity for pod assignment	`{}`
`bomengine.podSecurityContext`	BOM Engine security context at pod level	`{}`
`bomengine.securityContext`	BOM Engine security context at container level	`{}`

CFSSL pod configuration

Parameter	Description	Default
`cfssl.registry`	Image repository to be override at container level	
`cfssl.imageTag`	Image tag to be override at container level	`1.0.28`
`cfssl.resources.limits.memory`	Cfssl container Memory Limit	`640Mi`
`cfssl.resources.requests.memory`	Cfssl container Memory request	`640Mi`
`cfssl.persistentVolumeClaimName`	Point to an existing Cfssl Persistent Volume Claim (PVC)	
`cfssl.claimSize`	Cfssl Persistent Volume Claim (PVC) claim size	`2Gi`
`cfssl.storageClass`	Cfssl Persistent Volume Claim (PVC) storage class	
`cfssl.volumeName`	Point to an existing Cfssl Persistent Volume (PV)	
`cfssl.nodeSelector`	Cfssl node labels for pod assignment	`{}`

Parameter	Description	Default
`cfssl.tolerations`	Cfssl node tolerations for pod assignment	`[]`
`cfssl.affinity`	Cfssl node affinity for pod assignment	`{}`
`cfssl.podSecurityContext`	Cfssl security context at pod level	`{}`
`cfssl.securityContext`	Cfssl security context at container level	`{}`

Documentation pod configuration

Parameter	Description	Default
`documentation.registry`	Image repository to be override at container level	
`documentation.resources.limits.memory`	Documentation container Memory Limit	`512Mi`
`documentation.resources.requests.memory`	Documentation container Memory request	`512Mi`
`documentation.maxRamPercentage`	Documentation container Memory request	`90`
`documentation.nodeSelector`	Documentation node labels for pod assignment	`{}`
`documentation.tolerations`	Documentation node tolerations for pod assignment	`[]`
`documentation.affinity`	Documentation node affinity for pod assignment	`{}`
`documentation.podSecurityContext`	Documentation security context at pod level	`{}`
`documentation.securityContext`	Documentation security context at container level	`{}`

Integration pod configuration

Parameter	Description	Default
`integration.registry`	Image repository to be override at container level	
`integration.replicas`	Integration Pod Replica Count	`1`
`integration.resources.limits.cpu`	Integration container CPU Limit	`1000m`
`integration.resources.requests.cpu`	Integration container CPU request	`500m`
`integration.resources.limits.memory`	Integration container Memory Limit	`5120Mi`

Parameter	Description	Default
`integration.resources.requests.memory`	Integration container Memory request	`5120Mi`
`integration.maxRamPercentage`	Integration container maximum heap size	`90`
`integration.nodeSelector`	Integration node labels for pod assignment	`{}`
`integration.tolerations`	Integration node tolerations for pod assignment	`[]`
`integration.affinity`	Integration node affinity for pod assignment	`{}`
`integration.podSecurityContext`	Integration security context at pod level	`{}`
`integration.securityContext`	Integration security context at container level	`{}`

JDBC Connection Pool Configuration

Black Duck services use a JDBC connection pool to manage database connections. The default pool size is sufficient for most deployments. However, large enterprise environments with high API throughput (for example, 1000+ scans per hour) may need to increase the connection pool to avoid pool exhaustion errors.

Symptoms of an undersized connection pool

If the JDBC connection pool is too small for your workload, you may see errors like the following in your webapp pod logs:

```
org.apache.tomcat.jdbc.pool.PoolExhaustedException:
Timeout: Pool empty. Unable to fetch a connection in 30 seconds,
none available [size:32; busy:32; idle:0; lastwait:30000].
```

Other indicators include:

- HTTP API responses exceeding normal duration thresholds (e.g., 30 seconds or more for notification or project version endpoints).
- Webapp pods restarting frequently under sustained load.

Configuring `dbPoolMaxActive`

You can tune the maximum number of active JDBC connections for a given service by setting the `dbPoolMaxActive` property in your Helm chart `values.yaml` file under the appropriate service section.

Syntax:

```
<service>:
  dbPoolMaxActive: <integer>
```

Example — increasing the webapp connection pool to 64:

```
webapp:
  replicas: 1
  hubMaxMemory: "13824m"
```

```
  dbPoolMaxActive: 64
  resources:
    limits:
      cpu: "5000m"
      memory: "15360Mi"
    requests:
      cpu: "3000m"
      memory: "15360Mi"
```

The `dbPoolMaxActive` property can be set on any service that maintains a database connection pool (e.g., `webapp`, `jobrunner`, `bomengine`, `scan`).

Important considerations

Note: Tuning the connection pool is not required for most customers. The default values are appropriate for standard deployments. Only adjust this setting if you are experiencing pool exhaustion under sustained high-throughput workloads.

Before increasing `dbPoolMaxActive`, consider the following:

- **PostgreSQL** `max_connections`: Each active connection in the pool consumes a PostgreSQL backend connection. Ensure your PostgreSQL instance's `max_connections` setting can accommodate the total pool size across all services and replicas. For example, if you set `dbPoolMaxActive: 64` on `webapp` with 2 replicas, that service alone may use up to 128 PostgreSQL connections.
- **Database server resources:** Higher connection counts increase memory and CPU usage on the database server. Verify that your PostgreSQL host has sufficient resources.
- **Incremental changes:** Increase the pool size incrementally (e.g., from 32 → 64 → 96) and monitor the impact rather than making large jumps.
- **Holistic tuning:** Connection pool changes should be aligned with your overall API throughput requirements and PostgreSQL configuration. Revision should be managed with considered planning and caution.

Job runner pod configuration

Parameter	Description	Default
`jobrunner.registry`	Image repository to be override at container level	
`jobrunner.replicas`	Job runner Pod Replica Count	`1`
`jobrunner.resources.limits.cpu`	Job runner container CPU Limit	`1000m`
`jobrunner.resources.requests.cpu`	Job runner container CPU request	`1000m`
`jobrunner.resources.limits.memory`	Job runner container Memory Limit	`4608Mi`
`jobrunner.resources.requests.memory`	Job runner container Memory request	`4608Mi`
`jobrunner.maxRamPercentage`	Job runner container maximum heap size	`90`
`jobrunner.nodeSelector`	Job runner node labels for pod assignment	`{}`

Parameter	Description	Default
`jobrunner.tolerations`	Job runner node tolerations for pod assignment	`[]`
`jobrunner.affinity`	Job runner node affinity for pod assignment	`{}`
`jobrunner.podSecurityContext`	Job runner security context at pod level	`{}`
`jobrunner.securityContext`	Job runner security context at container level	`{}`

Logstash pod configuration

Parameter	Description	Default
`logstash.registry`	Image repository to be override at container level	
`logstash.imageTag`	Image tag to be override at container level	`1.0.38`
`logstash.resources.limits.memory`	Logstash container Memory Limit	`1024Mi`
`logstash.resources.requests.memory`	Logstash container Memory request	`1024Mi`
`logstash.maxRamPercentage`	Logsash maximum heap size	`90`
`logstash.persistentVolumeClaimName`	Point to an existing Logstash Persistent Volume Claim (PVC)	
`logstash.claimSize`	Logstash Persistent Volume Claim (PVC) claim size	`20Gi`
`logstash.storageClass`	Logstash Persistent Volume Claim (PVC) storage class	
`logstash.volumeName`	Point to an existing Logstash Persistent Volume (PV)	
`logstash.nodeSelector`	Logstash node labels for pod assignment	`{}`
`logstash.tolerations`	Logstash node tolerations for pod assignment	`[]`
`logstash.affinity`	Logstash node affinity for pod assignment	`{}`
`logstash.securityContext`	Logstash security context at container level	`{}`

Match engine pod configuration

Parameter	Description	Default
`matchengine.registry`	Image repository to be override at container level	
`matchengine.resources.limits.memory`	MATCH Engine container Memory Limit	`4608Mi`
`matchengine.resources.requests.memory`	MATCH Engine container Memory request	`4608Mi`
`matchengine.maxRamPercentage`	MATCH Engine maximum heap size	`90`
`matchengine.nodeSelector`	MATCH Engine node labels for pod assignment	`{}`
`matchengine.tolerations`	MATCH Engine node tolerations for pod assignment	`[]`
`matchengine.affinity`	MATCH Engine node affinity for pod assignment	`{}`
`matchengine.podSecurityContext`	MATCH Engine security context at pod level	`{}`
`matchengine.securityContext`	MATCH Engine security context at container level	`{}`

PostgreSQL pod configuration

Parameter	Description	Default
`postgres.registry`	Image repository	`docker.io/centos`
`postgres.isExternal`	If true, External PostgreSQL will be used	`true`
`postgres.host`	PostgreSQL host (required only if external PostgreSQL is used)	
`postgres.port`	PostgreSQL port	`5432`
`postgres.pathToPsqlInitScript`	Full file path of the PostgreSQL initialization script	`external-postgres-init.pgsql`
`postgres.ssl`	PostgreSQL SSL	`true`
`postgres.adminUserName`	PostgreSQL admin username	`postgres`
`postgres.adminPassword`	PostgreSQL admin user password	`testPassword`
`postgres.userUserName`	PostgreSQL non admin username	`blackduck_user`
`postgres.userPassword`	PostgreSQL non admin user password	`testPassword`

Parameter	Description	Default
`postgres.resources.requests.cpu`	PostgreSQL container CPU request (if external postgres is not used)	`1000m`
`postgres.resources.requests.memory`	PostgreSQL container Memory request (if external postgres is not used)	`3072Mi`
`postgres.persistentVolumeClaimName`	Point to an existing PostgreSQL Persistent Volume Claim (PVC) (if external postgres is not used)	
`postgres.claimSize`	PostgreSQL Persistent Volume Claim (PVC) claim size (if external postgres is not used)	`150Gi`
`postgres.storageClass`	PostgreSQL Persistent Volume Claim (PVC) storage class (if external postgres is not used)	
`postgres.volumeName`	Point to an existing PostgreSQL Persistent Volume (PV) (if external postgres is not used)	
`postgres.confPersistentVolumeClaimName`	Point to an existing PostgreSQL configuration Persistent Volume Claim (PVC) (if external postgres is not used)	
`postgres.confClaimSize`	PostgreSQL configuration Persistent Volume Claim (PVC) claim size (if external postgres is not used)	`5Mi`
`postgres.confStorageClass`	PostgreSQL configuration Persistent Volume Claim (PVC) storage class (if external postgres is not used)	
`postgres.confVolumeName`	Point to an existing PostgreSQL configuration Persistent Volume (PV) (if external postgres is not used)	
`postgres.nodeSelector`	PostgreSQL node labels for pod assignment	`{}`
`postgres.tolerations`	PostgreSQL node tolerations for pod assignment	`[]`
`postgres.affinity`	PostgreSQL node affinity for pod assignment	`{}`
`postgres.podSecurityContext`	PostgreSQL security context at pod level	`{}`

Parameter	Description	Default
`postgres.securityContext`	PostgreSQL security context at container level	`{}`

PostgreSQL readiness init container configuration

Parameter	Description	Default
`postgresWaiter.registry`	Image repository	
`postgresWaiter.podSecurityContext`	Postgres readiness check security context at pod level	`{}`
`postgresWaiter.securityContext`	Postgres readiness check context at container level	`{}`

PostgreSQL upgrade job configuration

Parameter	Description	Default
`postgresUpgrader.registry`	Image repository	
`postgresUpgrader.podSecurityContext`	Postgres upgrader security context at job level	`{}`
`postgresUpgrader.securityContext`	Postgres upgrader security context at container level	`{}`

RabbitMQ pod configuration

Parameter	Description	Default
`rabbitmq.registry`	Image repository to be override at container level	
`rabbitmq.imageTag`	Image tag to be override at container level	`1.2.40`
`rabbitmq.resources.limits.memory`	RabbitMQ container Memory Limit	`1024Mi`
`rabbitmq.resources.requests.memory`	RabbitMQ container Memory request	`1024Mi`
`rabbitmq.nodeSelector`	RabbitMQ node labels for pod assignment	`{}`
`rabbitmq.tolerations`	RabbitMQ node tolerations for pod assignment	`[]`
`rabbitmq.affinity`	RabbitMQ node affinity for pod assignment	`{}`
`rabbitmq.podSecurityContext`	RabbitMQ security context at pod level	`{}`
`rabbitmq.securityContext`	RabbitMQ security context at container level	`{}`

Redis pod configuration

Parameter	Description	Default
`redis.registry`	Image repository to be override at container level	
`redis.resources.limits.memory`	Redis container Memory Limit	`1024Mi`
`redis.resources.requests.memory`	Redis container Memory request	`1024Mi`
`redis.tlsEnalbed`	Enable TLS connections between client and Redis	`false`
`redis.maxTotal`	Maximum number of concurrent client connections that can be connected to Redis	`128`
`redis.maxIdle`	Maximum number of concurrent client connections that can remain idle in the pool, without extra ones being released	`128`
`redis.nodeSelector`	Redis node labels for pod assignment	`{}`
`redis.tolerations`	Redis node tolerations for pod assignment	`[]`
`redis.affinity`	Redis node affinity for pod assignment	`{}`
`redis.podSecurityContext`	Redis security context at pod level	`{}`
`redis.securityContext`	Redis security context at container level	`{}`

Registration pod configuration

Parameter	Description	Default
`registration.registry`	Image repository to be override at container level	
`registration.requestCpu`	Registration container CPU request	`1000m`
`registration.resources.limits.memory`	Registration container Memory Limit	`1024Mi`
`registration.resources.requests.memory`	Registration container Memory request	`1024Mi`
`registration.maxRamPercentage`	Registration container maximum heap size	`90`
`registration.persistentVolumeClaimName`	Point to an existing Registration Persistent Volume Claim (PVC)	

Parameter	Description	Default
`registration.claimSize`	Registration Persistent Volume Claim (PVC) claim size	`2Gi`
`registration.storageClass`	Registration Persistent Volume Claim (PVC) storage class	
`registration.volumeName`	Point to an existing Registration Persistent Volume (PV)	
`registration.nodeSelector`	Registration node labels for pod assignment	`{}`
`registration.tolerations`	Registration node tolerations for pod assignment	`[]`
`registration.affinity`	Registration node affinity for pod assignment	`{}`
`registration.podSecurityContext`	Registration security context at pod level	`{}`
`registration.securityContext`	Registration security context at container level	`{}`

Scan pod configuration

Parameter	Description	Default
`scan.registry`	Image repository to be override at container level	
`scan.replicas`	Scan Pod Replica Count	`1`
`scan.resources.limits.memory`	Scan container Memory Limit	`2560Mi`
`scan.resources.requests.memory`	Scan container Memory request	`2560Mi`
`scan.maxRamPercentage`	Scan container maximum heap size	`90`
`scan.nodeSelector`	Scan node labels for pod assignment	`{}`
`scan.tolerations`	Scan node tolerations for pod assignment	`[]`
`scan.affinity`	Scan node affinity for pod assignment	`{}`
`scan.podSecurityContext`	Scan security context at pod level	`{}`
`scan.securityContext`	Scan security context at container level	`{}`

Storage pod configuration

Parameter	Description	Default
`storage.registry`	Image repository to be override at container level	
`storage.requestCpu`	Storage container CPU request	`1000m`
`storage.resources.limits.memory`	Storage container Memory Limit	`2048Mi`
`storage.resources.requests.memory`	Storage container Memory request	`2048Mi`
`storage.maxRamPercentage`	Storage container maximum heap size	`60`
`storage.persistentVolumeClaimName`	Point to an existing Storage Persistent Volume Claim (PVC)	
`storage.claimSize`	Storage Persistent Volume Claim (PVC) claim size	`100Gi`
`storage.storageClass`	Storage Persistent Volume Claim (PVC) storage class	
`storage.volumeName`	Point to an existing Storage Persistent Volume (PV)	
`storage.nodeSelector`	Storage node labels for pod assignment	`{}`
`storage.tolerations`	Storage node tolerations for pod assignment	`[]`
`storage.affinity`	Storage node affinity for pod assignment	`{}`
`storage.podSecurityContext`	Storage security context at pod level	`{}`
`storage.securityContext`	Storage security context at container level	`{}`
`storage.providers`	Configuration to support multiple storage platforms. Please refer to *Storage Providers* section for additional details.	`[]`

Storage Providers

The provider in storage service refers to a persistence type and its configuration. Black Duck manages tools, application reports and other large blobs under storage service. Currently, it supports only the filesystem as one of the provider.

```
storage:
  providers:
    - name: <name-for-the-provider> <String>
      enabled: <flag-to-enable/disable-provider> <Boolean>
      index: <index-value-for-the-provider> <Integer>
      type: <storage-type> <String>
      preference: <weightage-for-the-provider> <Integer>
```

```
      existingPersistentVolumeClaimName: <existing-persistence-volume-claim-name> <String>
      pvc:
        size: <size-of-the-persistence-disk> <String>
        storageClass: <storage-class-name> <String>
        existingPersistentVolumeName: <existing-persistence-volume-name> <String>
      mountPath: <mount-path-for-the-volume> <String>
```

Parameter	Type	Description	Default
`name`	`String`	A name for the provider configuration. Eg. blackduck-file-storage	
`enabled`	`Boolean`	A flag to control enabling/ disabling of the provider instance	`false`
`index`	`Integer`	An index value for the provider configuration. Eg. 1,2,3.	
`type`	`String`	Storage type. Defaults to `file`.	`file`
`preference`	`Integer`	A number denoting the weightage for the provider instance configuration. If multiple provider instances are configured, then this value is used to determine which provider to be used as default storage option.	
`existingPersistentVolumeClaimName`	`String`	An option to re-use existing persistent volume claim for the provider	
`pvc.size`	`String`	The volume size to be used while creating persistent volume. A minimum size of `100Gi` is recommended for storage service.	`100Gi`
`pvc.storageClass`	`String`	The storage class to be used for persistent volume	
`pvc.existingPersistentVolumeName`	`String`	An option to re-use existing persistent volume for the provider	
`mountPath`	`String`	Path inside the container where the provider volume to be mounted	

Parameter	Type	Description	Default
`readonly`	`Boolean`	If present allows you to mark a provider as read only	false
`migrationMode`	`String`	Indicates if a migration is configured. Values can be 'NONE', DRAIN', 'DELETE' or 'DUPLICATE'	'NONE'

Webapp pod configuration

Parameter	Description	Default
`webapp.registry`	Image repository to be override at container level	
`webapp.resources.requests.cpu`	Webapp container CPU request	`1000m`
`webapp.resources.limits.memory`	Webapp container Memory Limit	`2560Mi`
`webapp.resources.requests.memory`	Webapp container Memory request	`2560Mi`
`webapp.maxRamPercentage`	Webapp container maximum heap size	`90`
`webapp.persistentVolumeClaimName`	Point to an existing Webapp Persistent Volume Claim (PVC)	
`webapp.claimSize`	Webapp Persistent Volume Claim (PVC) claim size	`2Gi`
`webapp.storageClass`	Webapp Persistent Volume Claim (PVC) storage class	
`webapp.volumeName`	Point to an existing Webapp Persistent Volume (PV)	
`webapp.nodeSelector`	Webapp node labels for pod assignment	`{}`
`webapp.tolerations`	Webapp node tolerations for pod assignment	`[]`
`webapp.affinity`	Webapp node affinity for pod assignment	`{}`
`webapp.podSecurityContext`	Webapp and Logstash security context at pod level	`{}`
`webapp.securityContext`	Webapp security context at container level	`{}`

Webserver pod configuration

Parameter	Description	Default
`webserver.registry`	Image repository to be override at container level	
`webserver.imageTag`	Image tag to be override at container level	`2024.7.1`
`webserver.resources.limits.memory`	Webserver container Memory Limit	`512Mi`
`webserver.resources.requests.memory`	Webserver container Memory request	`512Mi`
`webserver.nodeSelector`	Webserver node labels for pod assignment	`{}`
`webserver.tolerations`	Webserver node tolerations for pod assignment	`[]`
`webserver.affinity`	Webserver node affinity for pod assignment	`{}`
`webserver.podSecurityContext`	Webserver security context at pod level	`{}`
`webserver.securityContext`	Webserver security context at container level	`{}`

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.

Passing external database credentials via Kubernetes secret

When configuring Black Duck to use an external PostgreSQL database, you can choose to supply the database credentials via a Kubernetes secret rather than storing them directly in the `values.yaml` file. This approach enhances security by avoiding plaintext credentials in configuration files.

Using the default behavior (Helm-managed secret)

By default, the Helm chart will generate a secret named `<name>-blackduck-db-creds` using the values set from `adminPassword` and `userPassword` in your `values.yaml` file. This behavior is controlled by the `useHelmChartDbCreds` flag, which is enabled by default:

```
useHelmChartDbCreds: true
```

No additional steps are needed if you choose to continue using this method.

Providing your own database credentials secret

If you prefer to manage the credentials yourself, set `useHelmChartDbCreds` to `false` in your `values.yaml` file:

```
useHelmChartDbCreds: false
```

You must then create a Kubernetes secret named `<name>-blackduck-db-creds` in the same namespace as your Black Duck deployment. The secret must include the following keys:

- `HUB_POSTGRES_ADMIN_PASSWORD_FILE`
- `HUB_POSTGRES_USER_PASSWORD_FILE`

Each key should point to a file containing the corresponding password. For example:

```
kubectl create secret generic -n <namespace> <name>-blackduck-db-creds \
    --from-file=HUB_POSTGRES_ADMIN_PASSWORD_FILE=pg_admin_password_file \
    --from-file=HUB_POSTGRES_USER_PASSWORD_FILE=pg_user_password_file
```

Important: If the custom secret is invalid or missing, the deployment will fail. Helm will not fall back to using the credentials specified in `values.yaml`.

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`.

Setting	Details
	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. **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.

Setting	Details
	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.
`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

Migration Mode	Details
	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 blackduck-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.

Provisioning JWT public/private key pairs

To enhance the security and flexibility of JWT management, our system now supports the optional provisioning of public/private key pairs. This allows you to securely provide and manage these keys, ensuring they are only used by the appropriate services, such as the Authentication service for private keys and public API services for public keys.

Currently, only RSA keys (PEM encoded) are supported. Specifically, public keys must be in X.509 format, and private keys must be in PKCS#8 format.

Creating Kubernetes secrets

1. Create Kubernetes secret (template command). Exact files must be provided for the public and private key options.

   ```
   kubectl create secret generic -n <namespace> <name>-blackduck-jwt-keypair --from-
   file=JWT_PUBLIC_KEY=public_key_file --from-file=JWT_PRIVATE_KEY=private_key_file
   ```

   Here is an sample command if `namespace` = `bd` and `name` = `hub`:

   ```
   kubectl create secret generic -n bd hub-blackduck-jwt-keypair --from-
   file=JWT_PUBLIC_KEY=public-key.pem --from-file=JWT_PRIVATE_KEY=private-key.pem
   ```

2. Confirm secret created in `namespace`:

   ```
   kubectl get secrets -n <namespace>
   ```

   The expected output would be what follows, if `namespace` = `bd` and the secret name = `hub-blackduck-jwt-keypair`:

   ```
   kubectl get secrets -n bd
   NAME                        TYPE     DATA   AGE
   hub-blackduck-jwt-keypair   Opaque   2      7s
   ```

3. Uncomment the following line in `values.yaml` and edit `name` accordingly with the secret name:

   ```
   jwtKeyPairSecretName: <name>-blackduck-jwt-keypair
   ```

4. Deploy Black Duck in the same `namespace`. For example:

```
helm install bd . --namespace bd -f values.yaml -f sizes-gen04/10sph.yaml --set
 exposedNodePort=30000 --set environs.PUBLIC_HUB_WEBSERVER_PORT=30000
```

Enabling SCM Integration

This feature is not enabled by default in Black Duck and must be activated by adding the feature to your Product Registration key and then adding the following in your `values.yaml` file:

```
enableIntegration: true
```

Note: Black Duck does not accept self-signed certificates for SCM integrations at this time.

Configuring mTLS on an external database

Prerequisites

Before configuring mTLS for an external database in a Kubernetes deployment, ensure the following:

1. **Environment setup**:
   - Kubernetes and Helm are installed and properly configured.
   - Persistent Volumes (PV) and Persistent Volume Claims (PVC) are set up in your environment.
2. **Deployment preparation**:
   - You have identified the correct deployment files to use and know where to download them.
   - You are familiar with setting up a Black Duck deployment using an external database.
   - You have the necessary commands ready for running a Black Duck deployment with an external database.
3. **mTLS requirements**:
   - The `openssl.cnf` file is available on your server for generating certificates.
   - Secrets are named using the default values provided in the deployment instructions.
   - Depending on your deployment, you may have up to five secrets for mTLS: root certificate, admin certificate & key, and user certificate & key. If you are not using one or more of these, update the corresponding entries in the `values.yaml` file to an empty string ("").
     - Example: To configure the root certificate, update the `postgres.customCerts.rootCAKeyName` field in value.yaml. By default, this is set to "HUB_POSTGRES_CA". If you do not have a root certificate, set this value to "".
4. **Namespace consistency**:
   - Both the Black Duck and PostgreSQL deployments use the same namespace.

Changes to your Black Duck deployment

Updating `values.yaml`

To configure mTLS for your Black Duck deployment, you must update the `values.yaml` file with the necessary settings secrets. Follow these steps:

1. **Configure `postgres.sslMode`**

   Add the `postgres.sslMode` configuration option to `values.yaml`. This option determines whether the deployment will include certificate ane key secrets. If this option is not configured, the setenv.sh script will exclude the cert/key secrets from the deployment. Ensure this value is set to enable mTLS.

2. **Set the secret name for custom certificates**

   Add the `postgres.customCerts.useCustomCerts` option to `values.yaml`. This specifies the name of the secret containing the necessary certificate and key data for connecting to the external database (e.g., root CA, admin cert/key, and user cert/key).

3. **Define custom certificate data**

   Under the `postgres.customCerts` section in `values.yaml`, configure the following five options to specify the data for the certificates and keys:

   - `rootCAKeyName`: The key name for the root CA certificate.
   - `clientCertName`: The key name for the user certificate.
   - `clientKeyName`: The key name for the user private key.
   - `adminClientCertName`: The key name for the admin certificate.
   - `adminClientKeyName`: The key name for the admin private key.

   Ensure these values match the names in your secrets configuration.

Updating `postgres-init.yaml`

To support mTLS, the following changes are required in the `postgres-init.yaml` file:

1. Add volume and volume mount logic for the five new SSL secrets.
2. Include logic to set the `PGSSLMODE` environment variable.
3. Add logic to set the `PGSSLROOTCERT` environment variable.
4. Set the `PGSSLCERT` environment variable first based on `HUB_POSTGRES_CRT` and then on `HUB_ADMIN_POSTGRES_CRT`. Separate `if` statements should be used for clarity.
5. Set the `PGSSLKEY` environment variable first based on `HUB_POSTGRES_KEY` and then on `HUB_ADMIN_POSTGRES_KEY`. Separate `if` statements should be used for clarity.
6. Reorganize shell command logic by moving the logic for running `/tmp/postgres-init/init.pgsql` and grouping it with other shell commands for improved readability.

Update environs

Add `HUB_POSTGRES_ENABLE_SSL_CERT_AUTH: "true"` to environs.

Update `postgres.name`

After starting the database pod, retrieve the host value for the pod and update the `postgres.host` field in `values.yaml`.

Update postgres.customCerts.useCustomCerts

Update `postgres.customCerts.useCustomCerts` to `true`.

Create the secret for certificates and keys

Use the following command to create the secret for certificates and keys:

```
kubectl create secret generic -n bdbd-blackduck-postgres-certificate --from-
file=HUB_POSTGRES_CA=root.crt
        --from-file=HUB_POSTGRES_CRT=blackduck_user.crt
        --from-file=HUB_POSTGRES_KEY=blackduck_user.pk8
        --from-file=HUB_ADMIN_POSTGRES_CRT=blackduck_admin.crt
        --from-file=HUB_ADMIN_POSTGRES_KEY=blackduck_admin.pk8
```

Transitioning from Synopsysctl to helm chart deployments

As Black Duck evolves, we are transitioning from using synopsysctl to Helm charts for managing Kubernetes deployments. Helm charts provide a more standardized and flexible approach to deploying, upgrading, and maintaining Black Duck in Kubernetes environments.

In this guide, we recommend a fresh installation, as upgrading existing deployments may pose risks due to volume naming conventions and other configuration differences.

Important: Before beginning the transition process, ensure you back up your databases and any other critical data. This step is essential to prevent data loss in case of unexpected issues arising during the transition. Always verify the integrity of your backups before proceeding.

We strongly recommend deploying Black Duck in a test environment before proceeding with production. This allows you to validate the process and identify potential issues. The test environment can be a dedicated test instance or a temporary instance cloned from your production environment.

Internal or external database

If using an external database, you can configure the new installation to connect to the existing database, provided only one Black Duck instance communicates with it at any given time. Alternatively, you can create a new database and perform a backup and restore of your existing data.

If using an internal database, you must back up the current database and restore it to the new instance.

Transition process

1. Back up the database from the production environment.
2. Deploy a fresh installation of Black Duck using Helm in a new namespace.

   Note: If using an external database, configure the new installation to point to the existing external database during deployment.
3. Verify that the Black Duck instance is running correctly.
4. Restore the database using the production backup (for both internal or external databases).
5. Perform thorough testing to ensure functionality and data integrity.
6. If performing a dry-run, and it is successful, repeat the steps above after stopping the production instance and updating the DNS routing or load balancer to point to the new instance.