BLACKDUCK

BY SYNOPSYS®

Installing Black Duck using Kubernetes

Version 2019.2.2

This edition of the *Installing Black Duck using Kubernetes* refers to version 2019.2.2 of Black Duck.

This document created or updated on Tuesday, March 12, 2019.

Please send your comments and suggestions to:

Synopsys
800 District Avenue, Suite 201
Burlington, MA 01803-5061 USA

Copyright © 2019 by Synopsys.

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.

Contents

Chapter 1: Overview **1**
Black Duck Architecture 1
Components hosted on Black Duck servers 1
Chapter 2: Installation planning **2**
Getting started 2
New installations 2
Upgrading from a previous version of Black Duck 2
Hardware requirements 2
Kubernetes requirements 3
Operating systems 3
Software requirements 4
Network requirements 4
Additional port information 4
Database requirements 5
Understanding PostgreSQL's security configuration 5
Proxy server requirements 5
Configuring your NGiNX server to work with Black Duck 6
Amazon services 6
Configuring the keepalive setting 6
Chapter 3: Installing Black Duck **8**
Obtaining the orchestration files 8
Download from the GitHub page 9
Download using the wget command 9
Distributions 9
Using persistent volumes 10
Setting up persistent volumes 10
Creating a namespace 10
Enabling Black Duck with Black Duck - Binary Analysis 10
Customizing your Black Duck configuration files 11
Certificates 11
Installing a PostgreSQL database inside a container within the cluster 11
General Black Duck configuration 11
Installing Black Duck 12

Before you begin 12
Creating the service account, certificate service, and configuration map 13
Installing PostgreSQL inside a container 13
Using an external PostgreSQL database 14
Installing Black Duck - Binary Analysis 15
Creating Black Duck containers 15
Starting Over 15
Removing the Black Duck installation in Kubernetes 15
Connecting to Black Duck 15
Chapter 4: Administrative tasks 17
Understanding the default sysadmin user 17
Environment variables 17
Web server settings 18
Host name modification 18
Port modification 18
Disabling IPv6 18
Proxy settings 19
Proxy password 19
Managing certificates 19
Using a custom web server certificate-key pair in Kubernetes 20
Scaling containers 23
Scaling Job Runner containers 23
Scaling Scan containers 23
Scaling Binaryscanner containers 23
Configuring the report database password 23
Accessing the API documentation through a proxy server 24
Accessing the REST APIs from a non-Black Duck server 25
Configuring secure LDAP 25
Obtaining your LDAP information 26
Importing the server certificate 27
LDAP trust store password 28
Configuring SAML for Single Sign-On 28
Enabling the hierarchical BOM 31
Including ignored components in reports 31
Backing up PostgreSQL volumes 31
Increasing the size of the binary scan file 31
Configuring the containers' time zone 31
Modifying the default usage 32
Chapter 5: Upgrading Black Duck 34
Upgrading Black Duck on Kubernetes 34
Enabling Black Duck - Binary Analysis 34

Upgrading Black Duck 35
Backing up the PostgreSQL database 35
Upgrading the config map and containers 37
Recreating the webserver service 39
Improving external PostgreSQL database performance 40
Appendix A: Adding a persistent volume claim to a service 41
Creating a persistent volume claim for a service 41
Editing a configuration file for a service 42
Appendix B: Troubleshooting 44
Accessing log files 44
Purging logs 44
Appendix C: Containers 46
Authentication container 47
CA container 49
DB container 49
Documentation container 51
Jobrunner container 52
Logstash container 53
Registration container 53
Scan container 54
Solr container 55
Webapp container 56
WebServer container 57
ZooKeeper container 58
Black Duck - Binary Analysis containers 59
Binaryscanner container 59
Rabbitmq container 60
Uploadcache container 61

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 Compose	install_compose.pdf	Contains information about installing and upgrading Black Duck using Docker Compose.
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.

Title	File	Description
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.

Black Duck integration documentation can be found on Confluence.

Training

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

New videos and courses are added monthly.

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

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

Learn more at https://education.synopsys.com.

Customer Success Community

The Black Duck Customer Success Community is our primary online resource for customer support, solutions, and information. The Customer Success 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 Customer Success 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, please send an email to communityfeedback@blackducksoftware.com or call us at +1 781.891.5100 ext. 5.

To see all the ways you can interact with Black Duck Support, visit:

https://www.blackducksoftware.com/support/contact-support.

Chapter 1: Overview

Kubernetes is an orchestration tool used for managing cloud workloads through containers. This document provides instructions for installing Black Duck using Kubernetes.

Note: Synopsys recommends using Synopsys Operator to install and manage your Black Duck Kubernetes cluster. Click here for more information.

Black Duck Architecture

Black Duck is deployed as a set of containers so that third-party orchestration tools such as Kubernetes can be leveraged to manage individual Black Duck services.

This architecture brings these significant improvements to Black Duck over monolithic deployments:

- Improved performance
- Easier installation and updates
- Scalability
- Product component orchestration and stability

See containers for more information on the Docker containers that comprise the Black Duck application.

Visit the Kubernetes website for more information on Kubernetes.

Components hosted on Black Duck servers

The following remote Black Duck services are leveraged by Black Duck:

- Registration server: Used to validate Black Duck's license.
- Black Duck KnowledgeBase server: The Black Duck KnowledgeBase (KB) is the industry's most comprehensive database of open source project, license, and security information. Leveraging the Black Duck KB in the cloud ensures that Black Duck can display the most up-to-date information about open source software (OSS) without requiring regular updates to your Black Duck installation.

Chapter 2: Installation planning

This chapter describes the pre-installation planning and configuration that must be performed before you can install Black Duck.

Getting started

The process for installing Black Duck depends on whether you are installing Black Duck for the first time or upgrading from a previous version of Black Duck.

New installations

For new installation of Black Duck:

1. Read this planning chapter to review all requirements.
2. After ensuring that you meet all requirements, go to Chapter 3 for installation instructions.
3. Review Chapter 4 for any post-installation tasks.

Upgrading from a previous version of Black Duck

1. Read this planning chapter to review all requirements,
2. After ensuring that you meet all requirements, go to Chapter 5 for upgrade instructions.
3. Review Chapter 4 for any post-installation tasks.

Hardware requirements

The following is the minimum hardware that is needed to run a single instance of all containers:

- 5 CPU cores
- 20 GB RAM
- 250 GB of free disk space for the database and other Black Duck containers
- Commensurate space for database backups

The following is the minimum hardware that is needed to run Black Duck with Black Duck - Binary Analysis:

- 6 CPUs
- 24 GB RAM

- 350 GB of free disk space for the database and other Black Duck containers
- Commensurate space for database backups

Note: An additional CPU, 2 GB RAM, and 100 GB of free disk space is needed for every additional binaryscanner container.

The descriptions of each container provides the container's requirements, including if running on a different machine or if more than one instance of a container will be running (currently only supported for the job runner, scan, and binaryscanner containers).

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

In order to avoid underlying hardware resource exhaustion by Black Duck, ensure that your Kubernetes system administrator has put enterprise-level metrics and logging in place to identify unhealthy nodes on the cluster.

Kubernetes requirements

Black Duck supports Kubernetes versions 1.8.x through 1.10.x on Amazon Web Services (AWS) and Google Compute Engine (GCE)

If you are not using persistent volumes, these three restrictions apply when using Black Duck in Kubernetes:

- The PostgreSQL DB must always run on the same node in the cluster so that data is not lost (blackduck-database service).

  Storage must be provided for this node.

  This does *not* apply to installations using an external PostgreSQL instance.

- The webapp service and the logstash service must run on the same pod for proper log integration.

  This is required so that the webapp service can access the logs that need to be downloaded.

- The blackduck-registration service must always run on the same node in the cluster so that registration data is not lost.

  It does not need to be the same node as used for the blackduck-database service or the blackduck-webapp service.

Operating systems

The Dockerized Black Duck is supported on any Kubernetes cluster that passes the standards for Kubernetes cluster Conformance. (Click here for more on Kubernetes conformance.) Platforms that support Kubernetes include, but are not limited to:

- CentOS 7.3
- Red Hat Enterprise Linux server 7.3
- Ubuntu 16.04.x

- SUSE Linux Enterprise server version 12.x (64-bit)
- Oracle Enterprise Linux 7.3

Windows operating system is currently not supported.

Software requirements

Black Duck is a web application that has an HTML interface. You access the application via a web browser. The following web browser versions have been tested with Black Duck:

- Chrome 72.0.3626.96 (Official Build) (64-bit)
- Firefox 65.0 (64-bit)
- Internet Explorer 11.523.17134.0
- Microsoft Edge 42.17134.1.0
- Microsoft EdgeHTML 17.17134
- Safari 12.0 (14606.1.36.1.9)

Note that Black Duck does not support compatibility mode.

Note: These browser versions are the currently-released versions on which Black Duck Software has tested Black Duck. Newer browser versions may be available after Black Duck is released, and may or may not work as expected. Older browser versions may work as expected, but have not been tested and may not be supported.

Network requirements

Black Duck requires the following ports to be externally accessible:

- Port 443 – Web server HTTPS port for Black Duck via NGiNX
- Port 55436 – Read-only database port from PostgreSQL for reporting (or an equivalent exposable port for PostgreSQL read-only)

If your corporate security policy requires registration of specific URLs, connectivity from your Black Duck installation to Black Duck hosted servers is limited to communications via HTTPS/TCP on port 443 with the following servers:

- updates.suite.blackducksoftware.com (to register your software)
- kb.blackducksoftware.com (access the Black Duck KB data)

Note: If you are using a network proxy, these URLs must be configured as destinations in your proxy configuration.

Additional port information

The following list of ports cannot be blocked by firewall rules or by your Docker configuration. Examples of how these ports may be blocked include:

- The `iptables` configuration on the host machine.
- A `firewalld` configuration on the host machine.
- External firewall configurations on another router/server on the network.
- Special Docker networking rules applied above and beyond what Docker creates by default, and also what Black Duck creates by default.

The complete list of ports that must remain unblocked is:

- 443
- 8443
- 8000
- 8888
- 8983
- 16543
- 17543
- 16545
- 16544
- 55436

Database requirements

Black Duck uses the PostgreSQL object-relational database to store data.

For Black Duck version 2019.2.2, you must use PostgreSQL version 9.6.x for compatibility with Black Duck version 2019.2.2. Refer to Upgrading Black Duck for database migration instructions if upgrading from a pre-4.2.0 version of Black Duck.

Prior to installing Black Duck, determine whether you want to run PostgreSQL inside a container in a cluster or as an external PostgreSQL instance (for example, Amazon Relational Database Service (RDS)).

Understanding PostgreSQL's security configuration

PostgreSQL security is derived from CFSSL, which runs as a service inside your cluster.

For your Black Duck database to be secure, ensure that:

1. The namespace you are running PostgreSQL in is secure.
2. You have control over the users starting containers in that namespace.
3. The node which was labeled for PostgreSQL is protected from SSH by untrusted users.

Proxy server requirements

Black Duck supports:

- No Authentication
- Digest

- Basic
- NTLM

If you are going to make proxy requests to Black Duck, work with the proxy server administrator to get the following required information:

- The protocol used by proxy server host (http or https).
- The name of the proxy server host
- The port on which the proxy server host is listening.

Configuring your NGiNX server to work with Black Duck

Given that Kubernetes manages load balancing, there is no need to configure an NGiNX reverse proxy outside the external load balancer.

Amazon services

You can:

- Install Black Duck on Amazon Web Services (AWS)

  Refer to your AWS documentation for more information on AWS.

- Use Amazon Relational Database Service (RDS) for the PostgreSQL database that is used by Black Duck.

  Refer to your Amazon Relational Database Service documentation for more information on Amazon RDS.

  Currently Black Duck requires PostgreSQL version 9.6.x.

Configuring the keepalive setting

The `net.ipv4.tcp_keepalive_time` parameter controls how long an application will let an open TCP connection remain idle. By default, this value is 7200 seconds (2 hours).

For optimal Black Duck performance, this parameter should have a value between 600 and 800 seconds.

This setting can be configured before or after Black Duck is installed.

To edit the value

1. Edit the `/etc/sysctl.conf` file. For example:

   ```
   vi /etc/sysctl.conf
   ```

   You can also use the `sysctl` command to modify this file.

2. Add the `net.ipv4.tcp_keepalive_time` (if the parameter is not in the file), or edit the existing value (if the parameter is in the file).

   ```
   net.ipv4.tcp_keepalive_time = <value>
   ```

3. Save and exit the file.

4. Enter the following command to load the new setting:

   ```
   sysctl -p
   ```

5. If Black Duck is installed, restart it.

Chapter 3: Installing Black Duck

Prior to installing Black Duck, ensure that you meet the following requirements:

Black Duck Installation Requirements	
Hardware requirements	
☐	You have ensured that your hardware meets the minimum hardware requirements.
Kubernetes requirements	
☐	You have ensured that your system meets the Kubernetes requirements.
Software requirements	
☐	You have ensured that your system and potential clients meet the software requirements.
Network requirements	
☐	You have ensured that your network meets the network requirements. Specifically: • Port 443 and port 55436 are externally accessible. • The server has access to updates.suite.blackducksoftware.com which is used to validate the Black Duck license.
Database requirements	
☐	You have selected your database configuration.
Proxy requirements	
☐	You have ensured that your network meets the proxy requirements.
Web server requirements	
☐	Configure web server settings.

Obtaining the orchestration files

The installation files are available on Github (https://github.com/blackducksoftware/hub).

From your Kubernetes bastion host (host with access to the Internet and the Kube cluster) with kubectl installed, download the orchestration files. As part of the install/upgrade process, these orchestration files pull

down the necessary Docker images.

Note that although the filename of the `tar.gz` differs depending on how you access the file, the content is the same.

Download from the GitHub page

1. Select the link to download the `.tar.gz` file from the GitHub page: https://github.com/blackducksoftware/hub.
2. Uncompress the Black Duck `.gz` file:

   ```
   gunzip hub-2019.2.2.tar.gz
   ```

3. Unpack the Black Duck `.tar` file:

   ```
   tar xvf hub-2019.2.2.tar
   ```

Download using the wget command

1. Run the following command:

   ```
   wget https://github.com/blackducksoftware/hub/archive/v2019.2.2.tar.gz
   ```

2. Uncompress the Black Duck `.gz` file:

   ```
   gunzip v2019.2.2.tar.gz
   ```

3. Unpack the Black Duck `.tar` file:

   ```
   tar xvf v2019.2.2.tar
   ```

Distributions

The following is a list of files in the distribution for Kubernetes:

- `1-cfssl.yml`
- `1-cm-hub.yml`
- `2-binary-analysis.yml`
- `2-postgres-db-external.yml`
- `2-postgres-db-internal.yml`
- `3-hub.yml`
- `4-upgrade-prior-5.0.yml`
- `external-postgres-init.pgsql`

From the `bin` directory in the distribution:

- `hub_reportdb_changepassword.sh`: Script used to set and change the report database password.
- `hub_db_migrate.sh`: Script used to migrate the PostgreSQL database when using the database container provided by Black Duck.

Using persistent volumes

Both Black Duck's PostgreSQL database and various Black Duck components have data that must be stored. The default configuration files in the Black Duck installation media specify using `emptyDir` for storage, which provides temporary storage of data. This minimizes complexity, but could result in data loss if Black Duck containers are restarted and rescheduled.

For demonstrations and evaluations, using `emptyDir` is acceptable. In production environments, it is essential to modify Black Duck configuration to use persistent volume claims.

Setting up persistent volumes

Before Black Duck can be configured to use a persistent volume claim, you must first arrange for persistent volumes to be available in your cluster. A full discussion of the myriad ways to implement persistent storage (NFS, Gluster, and others) in a cluster environment is beyond the scope of this guide. Refer to the Red Hat OpenShift documentation on persistent volumes and persistent storage for more information. Consult your system administrator for assistance with storage in your cluster.

After persistent volumes are available, you can modify Black Duck configuration files to create persistent volume claims against the volumes. Do not proceed with a production Black Duck installation until persistent volumes are available in your cluster.

Note: PostgreSQL is known to have problems running in a container when writing to Gluster-based persistent volumes. If you are using Gluster for your underlying file system, Black Duck Software recommends using an external database. For additional information, refer to this documentation.

Creating a namespace

Create a virtual cluster, or namespace, for running Black Duck containers.

Any valid namespace will work, so long as it does not already exist on your cluster and you do not plan on running other applications in it: the namespace must be unique to Black Duck, in order to ensure proper service resolution.

For example:

```
kubectl create ns my-ns
```

The namespace ensures that all containers, spanning multiple nodes, within the namespace have the same DNS, config maps, and so on.

Enabling Black Duck with Black Duck - Binary Analysis

If you want to enable Black Duck - Binary Analysis, update the config map in the `1-cm-hub.yml` file. Change the value of USE_BINARY_UPLOADS to 1.

For example:

```
USE_BINARY_UPLOADS: "1"
```

Customizing your Black Duck configuration files

There are configuration files that must be customized before you can begin the installation of Black Duck.

The following steps refer to configuration files in the installation media you previously transferred to your bastion host.

Use the text editor of your choice to modify your configuration files in the following processes.

Certificates

If this is a production Black Duck installation, then it is highly recommended that you configure your Black Duck to leverage persistent volume claims for persistent storage. To do so, refer to the Adding a persistent volume claim to a Black Duck service. Then, make edits to `1-cfssl.yml` file with the data:

Service	CLAIM_NAME	STORAGE_SIZE	VOLUME_NAME
cfssl	pvclaim-bd-hub-cfssl	1Mi	dir-cfssl

Installing a PostgreSQL database inside a container within the cluster

Black Duck requires a PostgreSQL 9.6 database for all the Black Duck's data storage requirements. If installing the PostgreSQL database inside a container within the cluster:

1. The default database configuration references passwords in a secret called db-creds. You must now create that secret now. The command is:

   ```
   kubectl create secret generic db-creds --from-literal=blackduck=<admin_password> --from-literal=blackduck_user=<user_password> -n <namespace>
   ```

   replace `<admin_password>` and `<user_password>` with passwords of your choice.

2. If this is a production Black Duck installation, you must configure the database to use a persistent volume claim. To do so, refer to Adding a persistent volume claim to a Black Duck service. Then, make edits to the `2-postgres-db-external.yml` file using the following values:

Service	CLAIM_NAME	STORAGE_SIZE	VOLUME_NAME
postgres	pvclaim-bd-hub-postgres	250Gi	postgres-persistent-vol

General Black Duck configuration

1. If this is a production installation, Black Duck highly recommends that you configure your Black Duck to leverage persistent volume claims for persistent storage. If this is a non-production Black Duck, this step can be skipped. There are several Black Duck services that must be configured. To do so, refer to Adding a persistent volume claim to a Black Duck service. Then, make edits to the `3-hub.yml` file for each row in the table:

Service	CLAIM_NAME	STORAGE_SIZE	VOLUME_NAME
webapp	pvclaim-bd-hub-webapp	1Gi	dir-webapp
logstash	pvclaim-bd-hub-logstash	50Gi	dir-logstash
registration	pvclaim-bd-hub-registration	100Mi	dir-registration
zookeeper	pvclaim-bd-hub-zookeeper	1Gi	dir-zookeeper
authentication	pvclaim-bd-hub-authentication	100Mi	dir-authentication
rabbitmq*	pvclaim-bd-hub-rabbitmq	5 GB	dir-rabbitmq-data
uploadcache*	pvclaim-bd-hub-uploadcache	100 GB	dir-uploadcache-data
* Used for Black Duck - Binary Analysis only.			

Aside from these persistent volumes the 'binaryscanner' deployment will need 50-100GB of disk space that will not need to be saved or be persistent – it is just 'working' disk space. This is part of the additional disk space requirements for Black Duck - Binary Analysis.

1. If you are using a network proxy:

   a. These Black Duck services installed in the project (Authentication, Registration, Jobrunner, Webapp and Scan) must be configured to access the following URLs:
      - https://updates.suite.blackducksoftware.com
      - https://kb.blackducksoftware.com

   b. You must add the proxy environment variables into the `1-cm-hub.yml` file. The variables are:
      - HUB_PROXY_HOST. Name of the proxy server host.
      - HUB_PROXY_PORT. The port on which the proxy server host is listening.
      - HUB_PROXY_SCHEME. Protocol to use to connect to the proxy server.
      - HUB_PROXY_USER. Username to access the proxy server.

      For NTLM proxies, the variables are:
      - HUB_PROXY_WORKSTATION. The workstation the authentication request is originating from. Essentially, the computer name for this machine.
      - HUB_PROXY_DOMAIN. The domain to authenticate within.

Installing Black Duck

Now that your configuration files are customized to your environment, you can start the Black Duck deployment. The following commands must be run from the bastion host containing your configuration files.

Before you begin

Black Duck Software recommends that you make a backup copy of the configuration files that you previously edited. Using a version-control system is ideal, but other mechanisms are possible. Additionally, you may want

to run your edited configuration files through a YML syntax-verifier; for example, YAML Lint, to verify that you have not introduced syntax errors into the files.

Creating the service account, certificate service, and configuration map

Only users installing PostgreSQL inside a container within the cluster should create the service account:

```
kubectl create serviceaccount postgresapp -n <namespace>
```

All users need to create the certificate service and configuration map:

```
kubectl create -f 1-cfssl.yml -n <namespace>

kubectl create -f 1-cm-hub.yml -n <namespace>
```

Installing PostgreSQL inside a container

Go to the next section, Using an external PostgreSQL database, if using an external database with Black Duck. Otherwise, to install PostgreSQL inside a container within the cluster, run the command:

```
kubectl create -f 2-postgres-db-external.yml -n <namespace>
```

You now have a fresh deployment of PostgreSQL in your cluster. You can see the pod you created using the command:

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

Initializing PostgreSQL for use with Black Duck

Now that PostgreSQL is installed, it must be initialized with Black Duck-specific data. This section describes that initialization process.

1. Open the `external-postgres-init.pgsql` file in an editor.
2. Immediately after the line:

   ```
   CREATE USER blackduck_reporter;
   ```

   Add the following two lines:

   ```
   ALTER USER blackduck_user WITH password '<my_postgresql_password>';

   ALTER USER blackduck WITH password '<my_postgresql_admin_password>';
   ```

3. Save and exit the file.

   Verify that the blackduck_user password matches the POSTGRESQL_PASSWORD set in the `2-postgres-db-external.yml` file. Also, the blackduck password must match the POSTGRESQL_ADMIN_PASSWORD in the `2-postgres-db-external.yml` file.

4. Run the command:

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

   to find the pod name for the PostgreSQL database pod. For example:

```
NAME              READY   STATUS    RESTARTS   AGE
postgres-z846t    1/1     Running   0          57m
```

5. Copy the `external-postgres-init.pgsql` file into the database pod using the command:

   ```
   kubectl cp ./external-postgres-init.pgsql <namespace>/<pod-name>:external-postgres-init.pgsql
   ```

6. Run the command:

   ```
   kubectl exec -t -i <pod id> -n <namespace> -- /bin/sh
   ```

   to shell into the database container.

7. In the shell, run the command:

   ```
   psql -a -f external-postgres-init.pgsql
   ```

8. Exit the container by typing:

   ```
   exit
   ```

At this point, your database is initialized, and you are ready to install Black Duck.

Note: To use the reporting database in Black Duck, you must set the password for blackduck_reporter to enable that account. Use the same ALTER USER commands as previously described.

Using an external PostgreSQL database

Initialize the external PostgreSQL cluster with the "C" locale. The method to accomplish this depends on what your external PostgreSQL provider allows you to do. For example, when using the PostgreSQL initdb tool, run the following command:

```
initdb --locale=C -D /path/to/data
```

When using other tools, an equivalent alternative if a locale setting is not available is specifying the SQL_ASCII character encoding.

Run the following commands only if using an external PostgreSQL database.

1. Run the following command:

   ```
   kubectl create secret generic db-creds --from -literal=blackduck=<my_postgresql_admin_password> --from-literal=blackduck_user=<my_postgresql_password> -n <namespace>
   ```

2. On your external database, run these commands:

   ```
   psql -a -f /tmp/external-postgres-init.pgsql
   ```

   ```
   psql -c "ALTER USER blackduck_user WITH password '<my_postgresql_password>'"
   ```

```
psql -c " ALTER USER blackduck WITH password '<my_postgresql_admin_
password>'"
```

Installing Black Duck - Binary Analysis

If you plan to enable Black Duck - Binary Analysis, which is a separately licensed feature, you will have to create additional deployment services. Run the following command to create these items:

```
kubectl create -f 2-binary-analysis.yml -n <namespace>
```

Creating Black Duck containers

Now that your PostgreSQL database is configured, you can create Black Duck containers. To do so, run the command:

```
kubectl create -f 3-hub.yml -n <namespace>
```

It will take several minutes for all the pods to start. At any time, you can see the progress of the pod creation using the command:

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

If, after several minutes, all pods show a status of 'Running', then Black Duck is installed.

Starting Over

If you need to edit the `3-hub.yml` file and re-create Black Duck, the best course of action is to delete the pods created by the `3-hub.yml` file, and then recreate them by running the command:

```
kubectl delete -f 3-hub.yml -n <namespace>
```

Followed by the command:

```
kubectl create -f 3-hub.yml -n <namespace>
```

Removing the Black Duck installation in Kubernetes

If you want to remove the entire PostgreSQL/Black Duck installation, use the command:

```
kubectl delete ns <my-namespace>
```

Connecting to Black Duck

Once all containers for Black Duck are up, the web application for Black Duck will be exposed on port 443 to the Docker host. Be sure that you have configured the hostname and then you can access Black Duck by entering the following:

https://hub.example.com

The first time you access Black Duck, the Registration & End User License Agreement appears. You must accept the terms and conditions to use Black Duck.

Enter the registration key provided to you to access Black Duck.

Note: If you need to reregister, you must accept the terms and conditions of the End User License Agreement again.

Chapter 4: Administrative tasks

This chapter describes these administrative tasks:

- Understanding the default sysadmin user.
- Configuring web server settings, such as configuring the hostname, host port, or disabling IPv6.
- Configuring proxy settings.
- Replacing the existing self-signed certificate for the Web Server with a custom certificate.
- Scaling containers.
- Configuring the report database password.
- Providing access to the API documentation through a proxy server.
- Providing access to the REST APIs from a non-Black Duck server.
- Configuring secure LDAP.
- Configuring Single Sign-On (SSO).
- Enabling the hierarchical BOM.
- Including ignored components in reports
- Backing up PostgreSQL volumes.
- Increasing the size of the binary scan file
- Configuring the containers' time zone
- Modifying the default usage

Understanding the default sysadmin user

When you install Black Duck, there is a default system administrator (sysadmin) account already configured. The default sysadmin user has all roles and permissions associated with it.

Tip: As a best practice, you should use the default sysadmin account for your initial log in and then immediately change the default password—blackduck—so that the server is secure. To change your password, select **My Profile** from your username/user profile icon in the upper right corner of the Black Duck UI.

Environment variables

Several environment variables can be set to customize your Black Duck installation in a Kubernetes environment.

Note that If you wish to modify an environment variable setting *before* you install Black Duck, simply edit the `1-`

`cm-hub.yml` file appropriately, then run the "kubectl create" command. But if you wish to modify an environment variable *after* you have installed Black Duck, it is best to use the "kubectl edit" command.

Run the following command and replace "<my_ns>" with the name of your namespace:

```
kubectl edit cm hub-config -n <my_ns> -o yaml
```

Running this command displays the config map in a "vi" editor. Add the environment variable you wish to change.

Note: To edit the value, press "i" to edit, modify the field accordingly, then press ":wq" to save the config map and exit.

Web server settings

The following sections describe the required web server settings for a Kubernetes environment.

Host name modification

When the web server starts up, if it does not have certificates configured, a self-signed certificate is generated. To ensure that the hostname on the self-signed certificate matches the hostname actually used to reach the web server, you must set the web server hostname. Otherwise, the certificate uses the service name as the hostname, and SSL handshake errors could result.

To inform the webserver of the hostname used to reach it, edit the `1-cm-hub.yml` file to update the desired host name value.

PUBLIC_HUB_WEBSERVER_HOST=LOCALHOST value

Port modification

In a Kubernetes environment, it is common to leverage an External Load Balancer (ELB) to forward network requests to nodes. In a Black Duck installation in Kubernetes, this External Load Balancer will forward web traffic to Black Duck's NGiNX proxy server, which sends traffic along to Black Duck's webapp.

If you want to change either the port that external users use to connect to the web server (for example, a web browser connecting to the Black Duck's web UI), or, the port that the NGiNX proxy server listens on from the ELB, you need to update the `1-cm-hub.yml` file.

To change the publicly-exposed web server port, edit PUBLIC_HUB_WEBSERVER_PORT from its default value of 443.

To change the port that the NGiNX listens to from the ELB, edit HUB_WEBSERVER_PORT from its default value of 8443.

Disabling IPv6

By default, NGiNX listens on IPv4 and IPv6. If IPv6 is disabled on a host machine, change the value of the IPV4_ONLY value in the HUB WEBSERVER SECTION in the `1-cm-hub.yml` file to 1.

Proxy settings

These are the services requiring access to services hosted by Black Duck Software:

- Authentication
- Registration
- Jobrunner
- Webapp
- Scan

If a proxy is required for external internet access, you must configure it in the `1-cm-hub.yml` file.

Proxy environment variables are:

- HUB_PROXY_HOST. Name of the proxy server host.
- HUB_PROXY_PORT. The port on which the proxy server host is listening.
- HUB_PROXY_SCHEME. Protocol to use to connect to the proxy server.
- HUB_PROXY_USER. Username to access the proxy server.

The environment variables for NTLM proxies are:

- HUB_PROXY_WORKSTATION. The workstation the authentication request is originating from. Essentially, the computer name for this machine.
- HUB_PROXY_DOMAIN. The domain to authenticate within.

Proxy password

In the `1-cm-hub.yml` file specify the proxy password by entering it as the HUB_PROXY_PASSWORD value. Note that it must be a base64 encoded password.

```
# If you are using a proxy password, creation of this stanza will fail.
# that is ok.
- apiVersion: v1
  kind: Secret
    metadata:
      name: hub-proxy-pass
  data:
    HUB_PROXY_PASSWORD: "ZHVtbXkK"
```

Managing certificates

By default, Black Duck uses an HTTPS connection. The default certificate used to run HTTPS is a self-signed certificate which means that it was created locally and was not signed by a recognized Certificate Authority (CA).

If you use this default certificate, you will need to make a security exception to log in to Black Duck's UI, as your browser does not recognize the issuer of the certificate, so it is not accepted by default.

You will also receive a message regarding the certificate when connecting to the Black Duck server when

scanning as the scanner cannot verify the certificate because it is a self-signed and is not issued by a CA.

You can obtain a signed SSL certificate from a Certificate Authority of your choice. To obtain a signed SSL certificate, create a Certificate Signing Request (CSR), which the CA then uses to create a certificate that will identify the server running your Black Duck instance as "secure". After you receive your signed SSL certificate from the CA, you can replace the self-signed certificate.

To create an SSL certificate keystore

1. At the command line, to generate your SSL key and a CSR, type:

   ```
   openssl genrsa -out <keyfile> <keystrength>
   openssl req -new -key <keyfile> -out <CSRfile>
   ```

   where:

   - **<keyfile>** is <your company's server name>.key
   - **<keystrength>** is the size of your site's public encryption key
   - **<CSRfile>** is <your company's server name>.csr

   Note: It is important that the name entered for your company's server be the full hostname that your SSL server will reside on, and that the organization name be identical to what is in the 'whois' record for the domain.

   For example:

   ```
   openssl genrsa -out server.company.com.key 1024
   openssl req -new -key server.company.com.key -out server.company.com.csr
   ```

   This example creates a CSR for server.company.com to get a certificate from the CA.

2. Send the CSR to the CA by their preferred method (usually through a web portal).
3. Indicate that you need a certificate for an Apache web server.
4. Provide any requested information about your company to the CA. This information must match your domain registry information.
5. Once you receive your certificate from the CA, use the instructions in the next section to upload the certificate into a Black Duck instance.

Using a custom web server certificate-key pair in Kubernetes

You can use your own web server certificate-key pairs for establishing secure socket layer (SSL) connections to the Black Duck's web server.

1. To use a custom certificate, create a Kubernetes secret called `nginx-certs` with the custom certificate and custom key, respectively, in your namespace:

   ```
   kubectl create secret generic nginx-certs --from-file=WEBSERVER_CUSTOM_
   ```

```
CERT_FILE --from-file=WEBSERVER_CUSTOM_KEY_FILE -n <namespace>
```

2. In the `3-hub.yml` file, find the commented sections for the webserver certificate ("Uncomment this line to add a custom TLS Certificate for the web server.") and uncomment the lines.

```
      spec:
        volumes:
        - name: dir-webserver
          emptyDir: {medium: "Memory"}
        # Uncomment this line to add a custom TLS Certificate for
the web server.
        # - name: nginx-certs
        #   secret:
        #     secretName: nginx-certs
        #     items:
        #     - key: WEBSERVER_CUSTOM_CERT_FILE
        #       path: WEBSERVER_CUSTOM_CERT_FILE
        #     - key: WEBSERVER_CUSTOM_KEY_FILE
        #       path: WEBSERVER_CUSTOM_KEY_FILE
        containers:
        - name: webserver
          image: blackducksoftware/blackduck-nginx:<version>
          envFrom:
          - configMapRef:
              name: hub-config
          resources:
            requests:
              memory: 512M
            limits:
              memory: 512M
          livenessProbe:
            exec:
              command:
                - /usr/local/bin/docker-healthcheck.sh
                - https://localhost:8443/health-checks/liveness
                - /opt/blackduck/hub/webserver/security/root.crt
            initialDelaySeconds: 240
            timeoutSeconds: 10
            periodSeconds: 30
            failureThreshold: 10
          imagePullPolicy: Always
          ports:
          - containerPort: 8443
            protocol: TCP
          volumeMounts:
          - name: dir-webserver
            mountPath: "/opt/blackduck/hub/webserver/security"
          # Uncomment this line to add a custom TLS Certificate
for the web server.
          #- name: nginx-certs
          #  mountPath: "/tmp/secrets"
```

Scaling containers

The Job Runner, Scan, and Binaryscanner containers can be scaled up or down.

Scaling Job Runner containers

This example adds a second Job Runner container:

```
kubectl scale rc jobrunner --replicas=2
```

You can remove a Job Runner container by specifying a lower number than the current number of Job Runner containers. The following example scales back the Job Runner container to a single container:

```
kubectl scale rc jobrunner --replicas=1
```

Scaling Scan containers

This example adds a second Scan container:

```
kubectl scale rc scan --replicas=2
```

You can remove a Scan container by specifying a lower number than the current number of Scan containers. The following example scales back the Scan container to a single container:

```
kubectl scale rc scan --replicas=1
```

Scaling Binaryscanner containers

This example adds a second Binaryscanner container:

```
kubectl scale rc binaryscanner --replicas=2
```

Note: An additional CPU, 2 GB RAM, and 100 GB of free disk space is needed for every additional binaryscanner container.

You can remove a Binaryscanner container by specifying a lower number than the current number of Binaryscanner containers. The following example scales back the Binaryscanner container to a single container:

```
kubectl scale rc scan --replicas=1
```

Configuring the report database password

A PostgreSQL report database provides access to Black Duck data for reporting purposes. The database port is exposed to the Kuberentes network for connections to the report user and report database.

Note the following:

- Exposed port: 55436
- Username: blackduck_reporter. This user has read-only access to the database.
- Reporting database name: `bds_hub_report`
- Reporting user password. Not initially set.

- If using the database container that is automatically installed by Black Duck, use the provided script, as described below, to set the password before connecting to the database.
- If using an external PostgreSQL database, use your preferred PostgreSQL administration tool to configure the password.

Use the `hub_reportdb_changepassword.sh` script to set or change the report database password.

Note: This script sets or changes the report database password when using the database container that is automatically installed by Black Duck. If you are using an external postgreSQL database, use your preferred PostgreSQL administration tool to configure the password.

Note that to run the script to set or change the password:

- You may need to be a user in the docker group, a root user, or have `sudo` access.
- You must be on the Kubernetes node that is running the PostgreSQL database container.

In the following example, the report database password is set to 'blackduck':

```
./bin/hub_reportdb_changepassword.sh blackduck
```

After the password is set, you can connect to the reporting database.

For example, run the following command to obtain information about the internal and external IP address for your PostgreSQL service:

```
kubectl get service postgres -o wide
```

The command displays information such as the following:

```
NAME       CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
postgres   1.2.3.4         <none>        5432/TCP   9d
```

If your PostgreSQL client is inside the cluster, the external IP will be empty. If your PostgreSQL client is outside the cluster, take the external IP address and run the following command to open an interactive Postgres terminal to the remote database:

```
psql -U blackduck_reporter -p 55436 -h $external_ip_from_above -W bds_hub_
report
```

Accessing the API documentation through a proxy server

If you are using a reverse proxy and that reverse proxy has Black Duck under a subpath, configure the BLACKDUCK_SWAGGER_PROXY_PREFIX property so that you can access the API documentation. The value of BLACKDUCK_SWAGGER_PROXY_PREFIX is the Black Duck path. For example, if you have Black Duck being accessed under 'https://customer.companyname.com/hub' then the value of BLACKDUCK_SWAGGER_PROXY_PREFIX would be 'hub'.

To modify the property after installing Black Duck, add the environment variable above into the blackduck-nginx image stanza in the `3-hub.yml` file.

Accessing the REST APIs from a non-Black Duck server

You may wish to access Black Duck REST APIs from a web page that was served from a non-Black Duck server.

To enable this feature, Cross Origin Resource Sharing (CORS) must be enabled.

The properties used to enable and configure CORS for Black Duck installations are:

Property	Description
BLACKDUCK.HUB.CORS.ENABLED	Required. Defines whether CORS is enabled; "true" indicates CORS is enabled.
BLACKDUCK.CORS.ALLOWED.ORIGINS.PROP.NAME	Required. Allowed origins for CORS. The browser sends an origin header when it makes a cross-origin request. This is the origin that must be listed in the `blackduck.hub.cors.allowedOrigins` /`BLACKDUCK_CORS_ALLOWED_ORIGINS_PROP_NAME` property. For example, if you are running a server that serves a page from http:///123.34.5.67:8080, then the browser should set this as the origin, and this value should be added to the property. Note that the protocol, host, and port must match. Use a comma-separated list to specify more than one base origin URL.
BLACKDUCK.CORS.ALLOWED.HEADERS.PROP.NAME	Optional. Headers that can be used to make the requests.
BLACKDUCK.CORS.EXPOSED.HEADERS.PROP.NAME	Optional. Headers that can be accessed by the browser requesting CORS.

To modify the property after installing Black Duck, add the environment variables above into the blackduck-nginx image stanza in the `3-hub.yml` file.

Configuring secure LDAP

If you see certificate issues when connecting your secure LDAP server to Black Duck, the most likely reason is that the Black Duck server has not set up a trust connection to the secure LDAP server. This usually occurs if you are using a self-signed certificate.

To set up a trust connection to the secure LDAP server, import the server certificate into the local Black Duck LDAP truststore by:

1. Obtaining your LDAP information.
2. Using the Black Duck UI to import the server certificate.

Note: All hosted customers should secure access to their Black Duck application by leveraging our out-of-the-box support for single sign on (SSO) via SAML or LDAP. Information on how to enable and configure these security features can be found in the installation guides. In addition, we encourage customers that are using a SAML SSO provider that offers two-factor authorization to also enable and leverage that technology to further secure access to their Black Duck application.

Obtaining your LDAP information

Contact your LDAP administrator and gather the following information:

LDAP Server Details

This is the information that Black Duck uses to connect to the directory server.

- (required) The host name or IP address of the directory server, including the protocol scheme and port, on which the instance is listening.

  Example: `ldaps://<server_name>.<domain_name>.com:339`

- (optional) If your organization does not use anonymous authentication, and requires credentials for LDAP access, the password and either the LDAP name or the absolute LDAP distinguished name (DN) of a user that has permission to read the directory server.

  Example of an absolute LDAP DN: `uid=ldapmanager,ou=employees,dc=company,dc=com`

  Example of an LDAP name: `jdoe`

- (optional) If credentials are required for LDAP access, the authentication type to use: simple or digest-MD5.

LDAP Users Attributes

This is the information that Black Duck uses to locate users in the directory server:

- (required) The absolute base DN under which users can be located.

  Example: `dc=example,dc=com`

- (required) The attribute used to match a specific, unique user. The value of this attribute personalizes the user profile icon with the name of the user.

  Example: `uid={0}`

Test Username and Password

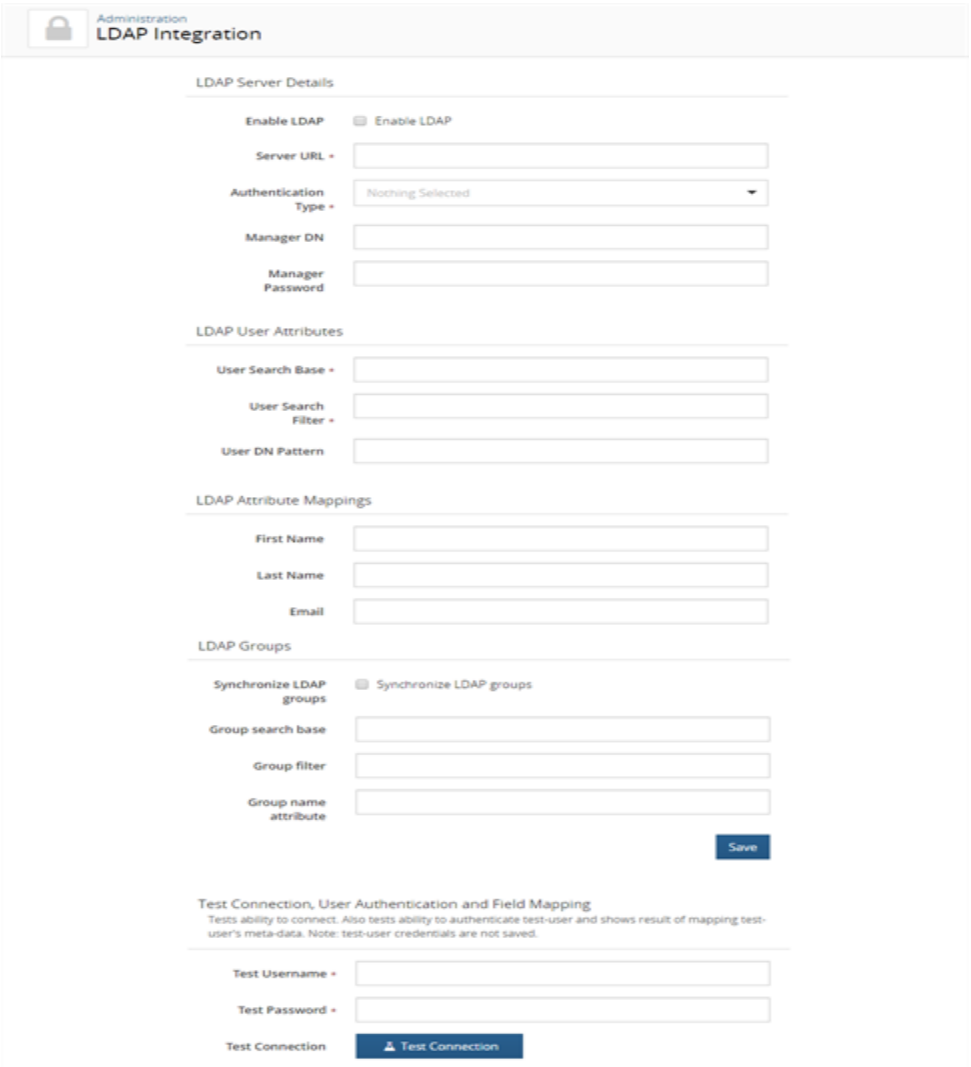
- (required) The user credentials to test the connection to the directory server.

Importing the server certificate

To import the server certificate

1. Log in to Black Duck as a system administrator.
2. Click the expanding menu icon ( ) and select **Administration**.

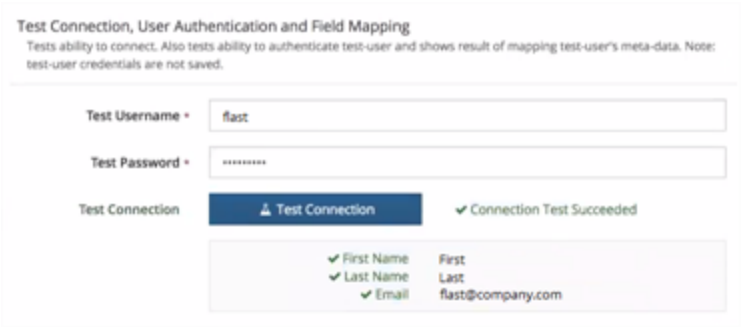
   The Administration page appears.
3. Select **LDAP integration** to display the LDAP Integration page.



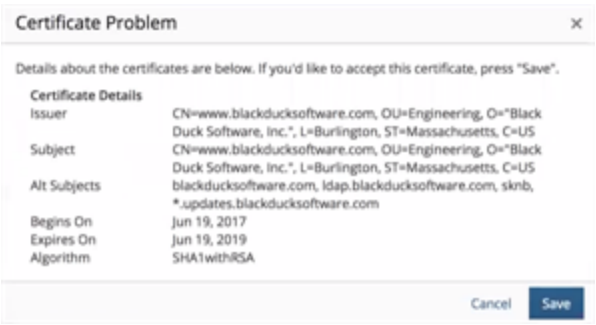
4. Select the **Enable LDAP** option and complete the information in the **LDAP Server Details** and **LDAP User Attributes** sections, as described above. In the **Server URL** field, ensure that you have configured the secure LDAP server: the protocol scheme is ldaps://.
5. Enter the user credentials in the **Test Connection, User Authentication and Field Mapping** section

and click **Test Connection**.

6. If there are no issues with the certificate, it is automatically imported and the "Connection Test Succeeded" message appears:



7. If there is an issue with the certificate, a dialog box listing details about the certificate appears:



Do one of the following:

- Click **Cancel** to fix the certificate issues.

  Once fixed, retest the connection to verify that the certificate issues have been fixed and the certificate has been imported. If successful, the "Connection Test Succeeded" message appears.

- Click **Save** to import this certificate.

  Verify that the certificate has been imported by clicking **Test Connection**. If successful, the "Connection Test Succeeded" message appears.

LDAP trust store password

For assistance in modifying an LDAP trust store password in a Kubernetes environment, contact your authorized Black Duck support representative.

Configuring SAML for Single Sign-On

Security Assertion Markup Language (SAML) is an XML-based, open-standard data format for exchanging authentication and authorization data between parties. For example, between an identity provider and a service provider. Black Duck's SAML implementation provides single sign-on (SSO) functionality, enabling Black Duck users to be automatically signed-in to Black Duck when SAML is enabled. Enabling SAML applies to all your

Black Duck users, and cannot be selectively applied to individual users.

Note: All hosted customers should secure access to their Black Duck application by leveraging our out-of-the-box support for single sign on (SSO) via SAML or LDAP. Information on how to enable and configure these security features can be found in the installation guides. In addition, we encourage customers that are using a SAML SSO provider that offers two-factor authorization to also enable and leverage that technology to further secure access to their Black Duck application.

To enable or disable SAML functionality, you must be a user with the system administrator role.

For additional SAML information:

- Assertion Consumer Service (ACS): https://host/saml/SSO
- Recommended Service Provider Entity ID: **https://host** where *host* is your Black Duck server location.

Note the following:

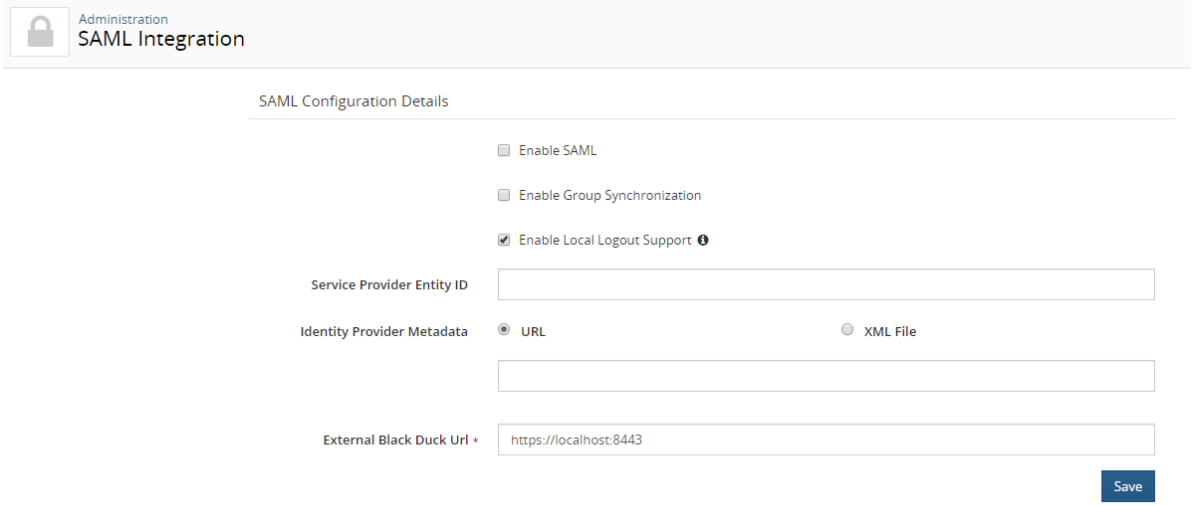
- Black Duck is able to synchronize and obtain an external user's information (Name, FirstName, LastName and Email) if the information is provided in attribute statements. Note that the first and last name values are case-sensitive.

  Black Duck is also able to synchronize an external user's group information if you enable group synchronization in Black Duck.
- When logging in with SAML enabled, you are re-directed to your identity provider's login page, not Black Duck's login page.
- When SSO users log out of Black Duck, a logout page now appears notifying them that they successfully logged out of Black Duck. This logout page includes a link to log back into Black Duck; users may not need to provide their credentials to successfully log back in to Black Duck.
- If there are issues with the SSO system and you need to disable the SSO configuration, you can enter the following URL: *Black Duck servername*/sso/login to log in to Black Duck.

To enable single sign-on using SAML

1. Click the expanding menu icon ( ) and select **Administration**.

   The Administration page appears.
2. Select **SAML Integration** to display the SAML Integration page.



3. In the **SAML Configuration Details** settings, complete the following:

   a. Select the **Enable SAML** check box.

   b. Optionally, select the **Enable Group Synchronization** check box. If this option is enabled, upon login, groups from the Identity Provider (IDP) are created in Black Duck and users will be assigned to those groups. Note that you must configure IDP to send groups in attribute statements with the attribute name of 'Groups'.

   c. Optionally, select the **Enable Local Logout Support** check box. If this option is enabled, after logging out of Black Duck, the IDP's login page would appear.

   Note: When local logout support is enabled, SAML requests are sent with ForceAuthn="true". Check with the IDP to confirm that this is supported.

   d. **Service Provider Entity ID** field. Enter the information for the Black Duck server in your environment in the format **https://*host*** where *host* is your Black Duck server.

   e. **Identity Provider Metadata**. Select one of the following:
      - **URL** and enter the URL for your identity provider.
      - **XML File** and either drop the file or click in the area shown to open a dialog box from which you can select the XML file.

   f. **External Black Duck Url** field. The URL of the public URL of the Black Duck server.

      For example: https://blackduck-docker01.dc1.lan

4. Click **Save**.

After clicking **Save**, the **BlackDuck Metadata URL** field appears. You can copy the link or directly download the SAML XML configuration information.

To disable single sign-on using SAML

1. Click the expanding menu icon ( ) and select **Administration**.
2. Select **SAML Integration** to display the SAML Integration page.
3. In the **SAML Configuration Details** settings, clear the **Enable SAML** check box.
4. Click **Save**.

Enabling the hierarchical BOM

By default, the hierarchical BOM is disabled. To enable this feature, add the HUB_HIERARCHICAL_BOM environment variable. Set the value to "true", for example, HUB_HIERARCHICAL_BOM=true.

Resetting the value to "false" disables the feature.

Including ignored components in reports

By default, ignored components and vulnerabilities associated with those ignored components are excluded from the Vulnerability Status report, Vulnerability Update report, Vulnerability Remediation report and the Project Version report. To include ignored components, set the value of the BLACKDUCK_REPORT_IGNORED_COMPONENTS environment variable to "true".

Resetting the value of the BLACKDUCK_REPORT_IGNORED_COMPONENTS to "false" excludes ignored components.

Backing up PostgreSQL volumes

Ensure that the volumes you use for PostgreSQL data storage are backed up on a regular basis. Consult your Kubernetes/Docker/PostgreSQL system administrator for information on how to back up PostgreSQL data volumes.

Increasing the size of the binary scan file

When using Black Duck - Binary Analysis, the maximum size of the binary that can be scanned is 6 GB. You can increase this limit by adding the environment variable BINARY_UPLOAD_MAX_SIZE and specifying a value in megabytes.

For example, to increase the maximum binary scan to 7 GB, add the following:

```
BINARY_UPLOAD_MAX_SIZE=7168m
```

Configuring the containers' time zone

By default, the time zone for Black Duck containers is UTC. For monitoring purposes, you may want to change this value so that the timestamps shown in logs reflect the local time zone.

To configure a different time zone, add the TZ environment variable and use the values shown in Wikipedia, as

shown here and restart the containers.

For example, to change the timezone to that used in Denver, Colorado, enter:

```
TZ=America/Denver
```

Modifying the default usage

Usage indicates how a component is intended to be included in the project when this version is released.

Possible usage values are:

- Statically Linked. A tightly-integrated component that is statically linked in and distributed with your project.
- Dynamically Linked. A moderately-integrated component that is dynamically linked in, such as with DLLs or .jar files.
- Source Code. Source code such as .java or .cpp files.
- Dev Tool / Excluded. Component will not be included in the released project. For example, a component that is used internally for building, development, or testing. Examples are unit tests, IDE files, or a compiler.
- Separate Work. Intended for loosely-integrated components. Your work is not derived from the component. To be considered a separate work, your application has its own executables, with no linking between the component and your application. An example is including the free Acrobat PDF Viewer with your distribution media.
- Implementation of Standard. Intended for cases where you implemented according to a standard. For example, a Java spec request that ships with your project.

The default usage is determined by match type: Snippets have a usage of Source Code while all other match types are Dynamically Linked.

Black Duck uses the following variables so that you can change the default usage for similar match types:

- BLACKDUCK_HUB_FILE_USAGE_DEFAULT. Defining a usage for this variable sets the default value for the following match types:
  - Binary
  - Exact Directory
  - Exact File
  - Files Added/Deleted
  - Files Modified
  - Partial
- BLACKDUCK_HUB_DEPENDENCY_USAGE_DEFAULT. Defining a usage for this variable sets the default value for the following match types:
  - File Dependency
  - Direct Dependency
  - Transitive Dependency
- BLACKDUCK_HUB_SOURCE_USAGE_DEFAULT. Defining a usage for this variable sets the default value

for the following match types:

- Snippet

- BLACKDUCK_HUB_MANUAL_USAGE_DEFAULT. Defining a usage for this variable sets the default value for the following match types:
  - Manually Added
  - Manually Identified

To configure different usage values

1. Add the environment variable. You *must* use the following text for the usage values: SOURCE_CODE, STATICALLY_LINKED, DYNAMICALLY_LINKED, SEPARATE_WORK, IMPLEMENTATION_OF_STANDARD, DEV_TOOL_EXCLUDED,

   For example, to change default usage for files to statically linked:

   ```
   BLACKDUCK_HUB_FILE_USAGE_DEFAULT=STATICALLY_LINKED
   ```

   Note: If you enter the incorrect usage text, the original default value will still apply. A warning message will appear in the log files of the jobrunner container.

The modified usage values apply to any new scans or rescans.

Chapter 5: Upgrading Black Duck

This chapter describes how to upgrade an existing Black Duck on Kubernetes to a newer version of Black Duck on Kubernetes.

Note: Upgrading Black Duck from a non-Kubernetes Black Duck installation (for example, AppMgr Black Duck) to Kubernetes is simply a fresh Black Duck install on Kubernetes plus a data migration. See Chapter 3 for information on fresh Black Duck installs.

Note: When upgrading from a version prior to 2018.12.0, you will experience a longer than usual upgrade time due to a data migration that is needed to support new features in this release. Upgrade times will depend on the size of the Black Duck database. If you would like to monitor the upgrade process, please contact Synopsys Customer Support for instructions.

Upgrading Black Duck on Kubernetes

Kubernetes applications can be upgraded using native Kubernetes image update commands. As such, upgrading Black Duck on Kubernetes is basically upgrading Black Duck's deployments (pods, essentially).

Your upgrade path depends on whether you are upgrading Black Duck and enabling Black Duck - Binary Analysis or just upgrading Black Duck:

- To upgrade *and* enable Black Duck - Binary Analysis, follow the instructions in the next section to replace the config map and enable Black Duck - Binary Analysis. Then follow the instructions described here to upgrade Black Duck.
- To Just upgrade Black Duck, follow the instructions described here to upgrade Black Duck which consists of:
  - Backing up the database.
  - Upgrading the config map.
  - Upgrading the containers.
  - Recreating the webserver service.

Enabling Black Duck - Binary Analysis

To enable Black Duck - Binary Analysis, you must replace the config map and then enable Black Duck - Binary Analysis.

Replacing the config map

If you have a config map created from a previous installation of Black Duck, you will need to replace it with a

new version of the file as there is a new property/value that enables Black Duck - Binary Analysis. Run the following command:

```
kubectl replace -f 1-cm-hub.yml -n <namespace>
```

If you have made any modifications to this file in previous installations, you will need to redo them.

Enabling Black Duck - Binary Analysis

To enable Black Duck - Binary Analysis, run the following command:

```
kubectl create -f 2-binary-analysis.yml -n <namespace>
```

Continue with the remaining steps listed below to upgrade Black Duck.

Upgrading Black Duck

Follow these procedures to upgrade Black Duck.

Upgrading Black Duck consists of:

1. Backing up the PostgreSQL database.
2. Upgrading the config map.
3. Upgrading the containers.
4. Recreating the webserver service.
5. Improve external PostgreSQL database performance.

Backing up the PostgreSQL database

Black Duck Software recommends completing a PostgreSQL database backup prior to upgrading Black Duck.

This section describes the process of backing up and restoring the Black Duck database data. This section covers:

- Backing up AppMgr Black Duck data (for migration purposes)
- Backing up Black Duck Kubernetes PostgreSQL data
- Restoring Black Duck Kubernetes PostgreSQL data

Note: In the instructions shown for backing up and restoring Kubernetes PostgreSQL data, for simplicity, a namespace is not declared. Please add a command line option such as `--namespace=my-ns` to every command shown below based on your administrator's conventions. If you do not do declare a namespace, the Black Duck containers will still work, however, they will all be created in the default namespace.

Backing up a PostgreSQL database from an AppMgr architecture

If you have a version of Black Duck using AppMgr whose data you want to migrate to a new Kubnernetes PostgreSQL node, follow these steps to back up the data.

To back up the original PostgreSQL database

1. Log in to Black Duck server as the **blckdck** user.

   Note: This is the user that owns Black Duck database and installation directory.

2. Run the following commands to dump to a compressed file.

   ```
   export PATH=$PATH:/opt/blackduck/hub/postgresql/bin
   export PGPORT=55436
   pg_dump -Fc -f /tmp/bds_hub.dump bds_hub
   ```

   Tip: Ensure that you dump the database to a location with sufficient free space. This example uses `/tmp`.

   This command puts the information from the `bds_hub` database into a file called `bds_hub.dump` in the `/tmp` directory. It ignores several scratch tables that do not need to be backed up.

3. Save the `bds_hub.dump` file on another system or offline.

Tip: If you find that dumping the database takes too long, you can greatly increase the speed by dumping it to an uncompressed file. The trade-off is that while the dump is completed up to 3 times faster, the resulting file may be 4 times larger. To experiment with this on your system, add the `--compress=0` parameter to your `pg_dump` command.

After completing these steps, go to Restoring/migrating database data.

Backing up a Kubernetes PostgreSQL database

To back up the Kubernetes PostgreSQL Black Duck database (the one that comes standard with Kubernetes Black Duck), you must locate the PostgreSQL node, SSH into it, and run a script that creates local backup files.

1. Find the node that is running PostgreSQL by running the following command:

   ```
   kubectl get nodes -l blackduck.hub.postgres=true
   ```

   Alternatively, you can get this information by doing a query such as the following:

   ```
   kubectl get pod postgres -o=jsonpath='{.spec.nodeName}'
   ```

   Note: The instructions in Step 1 show how to find the node that PostgreSQL is running on in Kubernetes. If you are using a different orchestration tool, use an equivalent command to find the hostname of the node, then go to Step 2.

2. Now that you know the hostname where Postgres is running, you must SSH into the node and run this command:

   Run the `hub_create_data_dump.sh` script. This script creates PostgreSQL data backup files in the blackduck-postgres container and then copies the files from the container to a local directory.

Important: You *must* run the 2019.2.2 version of the `hub_create_data_dump.sh` script located in the `kubernetes/bin` directory.

```
./bin/hub_create_data_dump.sh <local directory to store PostgreSQL data files>
```

This script will create a number of data backup files (`globals.sql`, `bds_hub.dump`, `bds_hub_report.dump`, and `bdio.dump`).

Restoring/migrating database data

Note: As mentioned previously, for each of the "kubectl" commands, below, make sure to include --namespace if required by your environment.

To restore your data from an existing database dump file:

1. Find the node that is running PostgreSQL by running the following command:

   ```
   kubectl get nodes -l blackduck.hub.postgres=true
   ```

   Alternatively, you can get this information by doing a query such as the following:

   ```
   kubectl get pod postgres -o=jsonpath='{.spec.nodeName}'
   ```

2. Now that you know the hostname where PostgreSQL is running, SSH into the node and run this command:

   ```
   ./hub_db_migrate.sh <local directory to load PostgreSQL data files>
   ```

   Important: You *must* run the 2019.2.2 version of the `hub_db_migrate.sh` script located in the `kubernetes/bin` directory.

Error messages

When the dump file is restored from the an AppMgr installation of Black Duck, you may receive error messages such as:

"ERROR: role "blckdck" does not exist"

along with other error messages. Also, at the end of the migration, you may see the following:

WARNING: errors ignored on restore: 7

These error messages and warnings can be ignored. They will not affect the restoration of the data.

Upgrading the config map and containers

Note: Black Duck Software recommends that no scans be active/initiated and that users remained logged off of the Black Duck web UI while the upgrade is occurring.

There are two steps to upgrading Black Duck in Kubernetes:

1. Upgrade the config map.
2. Upgrade the containers.

Note: If you are upgrading from a pre-4.8.0 version to 4.8.0 or later, and you use the default Postgres database in a container, then before proceeding, backup your database using the instructions in the previous section.

Upgrading the config map

1. To upgrade the config map, run the "kubectl edit" command to edit the config map in YAML format. Replace "<my_ns>" with the name of your namespace:

   ```
   kubectl edit cm hub-config -n <my_ns> -o yaml
   ```

   Running this command brings up the config map in a "vi" editor.

2. Search for "HUB_VERSION", and change the value to the version of Black Duck you are upgrading to.

Note: To edit the value, press "i" to edit, modify the version field accordingly, then press ":wq" to save the config map and exit.

Upgrading the containers

The command to upgrade a container in Kubernetes is:

```
kubectl set image <image> container_name=blackduck-image:version -n <my_ns>
```

The following Black Duck containers each needs to be individually updated:

- blackduck-cfssl
- blackduck-documentation
- blackduck-postgres
- blackduck-jobrunner
- blackduck-webapp
- blackduck-webserver
- blackduck-logstash
- blackduck-registration
- blackduck-solr
- blackduck-zookeeper
- blackduck-scan
- blackduck-authentication

For example, here are the specific commands that must be run to upgrade to Black Duck 2019.2.2:

```
kubectl set image deployment/cfssl
cfssl=docker.io/blackducksoftware/blackduck-cfssl:1.0.0 -n <my_ns>

kubectl set deployment/documentation
```

```
documentation=docker.io/blackducksoftware/blackduck-documentation:2019.2.2
-n <my_ns>

kubectl set image deployment/jobrunner
jobrunner=docker.io/blackducksoftware/blackduck-jobrunner:2019.2.2 -n <my_
ns>

kubectl set image deployment/postgres
postgres=docker.io/blackducksoftware/blackduck-postgres:1.0.1 -n <my_ns>
```

Note: This postgres command can be skipped if you use an external database.

```
kubectl set image deployment/webapp-logstash
webapp=docker.io/blackducksoftware/blackduck-webapp:2019.2.2 -n <my_ns>

kubectl set image deployment/webapp-logstash
logstash=docker.io/blackducksoftware/blackduck-logstash:1.0.2 -n <my_ns>

kubectl set image deployment/webserver
webserver=docker.io/blackducksoftware/blackduck-nginx:1.0.1 -n <my_ns>

kubectl set image deployment/registration
registration=docker.io/blackducksoftware/blackduck-registration:2019.2.2 -n
<my_ns>

kubectl set image deployment/solr
solr=docker.io/blackducksoftware/blackduck-solr:1.0.0 -n <my_ns>

kubectl set image deployment/zookeeper
zookeeper=docker.io/blackducksoftware/blackduck-zookeeper:1.0.0 -n <my_ns>

kubectl set image deployment/scan
scan=docker.io/blackducksoftware/blackduck-scan:2019.2.2 -n <my_ns>

kubectl set image deployment/authentication
authentication=docker.io/blackducksoftware/blackduck-
authentication:2019.2.2 -n <my_ns>
```

Note: If you are upgrading from a pre-4.8.0 version to 4.8.0 or later, and you use the default Postgres database in a container, then as a final step, restore your database backup using the procedures described in the previous section "Restoring/migrating database data".

Recreating the webserver service

There is an updated version of the webserver service in version 5.0 and later.

If you are upgrading from a version prior to 5.0.0 and already have a Black Duck installation and are using the `set-image` upgrade path, as described above, you will need to delete and recreate the webserver service by running these two commands:

```
kubectl delete service webserver -n <namespace>

kubectl create -f 4-upgrade-prior-5.0.yml -n <namespace>
```

Note that these commands can be run even if all of the deployments and services are already up.

Improving external PostgreSQL database performance

1. If you are upgrading from a pre-2018.12.1 version of Black Duck and are using an external PostgreSQL database, do the following:

   a. Using your preferred PostgreSQL administration tool, make these global system changes:

   ```
   autovacuum_max_workers = 20
   autovacuum_vacuum_cost_limit = 2000
   ```

   b. Restart PostgreSQL.

Appendix A: Adding a persistent volume claim to a service

This appendix describes how to request persistent storage for a particular Black Duck service using a Persistent Volume Claim.

Note: Before following these instructions, your cluster must have persistent volumes available against which to make claims. Click here for more information.

Prior to executing this procedure, you must know the following information:

- The name of the configuration file you are editing, for example, `1-cfssl.yml`.
- The name of the service whose stanza you are editing; for example, cfssl.

You must also know the following values, which you must substitute into the configuration file:

- CLAIM_NAME: The name of the persistent volume claim; for example, pvclaim-bd-hub-cfssl.
- STORAGE_SIZE: The amount of storage to request from the persistent volume; for example, 1Mi.
- VOLUME_NAME: The name of the persistent volume inside the pod; for example, pv-bd-hub-cfssl.

Do not proceed with these instructions until you have the required information.

Creating a persistent volume claim for a service

1. A separate persistent volume and persistent volume claim is required for each service for which you want to set up persistent storage. Assuming you have already created the persistent volume for a service, its corresponding persistent volume claim must have the following base form:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: CLAIM_NAME
  annotations:
    volume.beta.kubernetes.io/storage-class: ""
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: STORAGE_SIZE
```

Note: Your `accessMode` may be different, depending on your setup. Refer to the previous section for replacement values for CLAIM_NAME and STORAGE_SIZE.

2. After you have saved and created your persistent volume claim, double-check that the persistent volume claim is bound to the correct persistent volume. For example, the persistent volume claim for the cfssl service must be bound to the persistent volume for the cfssl service. If it is not, this must be addressed before continuing. A method for ensuring this is by utilizing volume and claim pre-binding. Refer to your OpenShift documentation for more information.

Editing a configuration file for a service

1. In the configuration file for the service for which you are setting up persistent storage, search for the stanza corresponding to that service. For example, if you are editing the cfssl service, then search for cfssl. Inside the service's stanza, there should be a volume reference with an `emptyDir` specification of the form:

```
volumes:
- emptyDir: {}
  name: VOLUME_NAME
```

Replace `emptyDir:{}` with the persistent volume claim information. The resulting stanza should have the form:

```
volumes:
- persistentVolumeClaim:
    claimName: CLAIM_NAME
  name: VOLUME_NAME
```

Refer to the previous section for replacement values for VOLUME_NAME and CLAIM_NAME.

2. In this same service stanza in your configuration file, there should be a volume mount stanza of the form:

```
volumeMounts:
- mountPath: /xxxx/yyyy/zzzz
  name: VOLUME_NAME
```

Ensure that the VOLUME_NAME in the volume mount section matches the VOLUME_NAME in the previous step. These values must match, as it is this common VOLUME_NAME that associates the claim with the mount.

Note: Do not change the `mountPath`. Each Black Duck service expects a particular mount path and is it already correctly specified.

3. Save and close the configuration file.

If a particular Black Duck pod fails to start, you can investigate the cause with the following command:

```
kubectl describe pod <pod-name> --namespace=<namespace-name>
```

View the **Events** section of the output. Causes and messages display; for example, a reason such as `FailedScheduling`, along with a message such as 'Insufficient memory'. In the case of insufficient resources, you can diagnose the issue by running the commands:

```
kubectl get nodes
```

and

```
kubectl describe node <node address>
```

These commands display the requests being made of the cluster by the node.

Accessing log files

You may need to troubleshoot an issue or provide log files to Customer Support.

Users with the System Administrator role can download a zipped file that contains the current log files.

To download the log files from the Black Duck UI

1. Log in to Black Duck with the System Administrator role.
2. Click the expanding menu icon ( ) and select **Administration**.

   The Administration page appears.
3. Select **System Settings**.

   The System Settings page appears.
4. Click **Download Logs (.zip)**.

   It may take a few minutes to prepare the log files.

Purging logs

Be default, log files are automatically purged after 30 days. To modify this value, add the DAYS_TO_KEEP_

LOGS environment variable and specify the number of days to keep files. For example, to purge the logs after 15 days, set DAYS_TO_KEEP_LOGS=15.

Click here for information on adding/editing environment variables.

Appendix C: Containers

These are the containers within the Docker network that comprise the Black Duck application:

1. Authentication
2. CA
3. DB

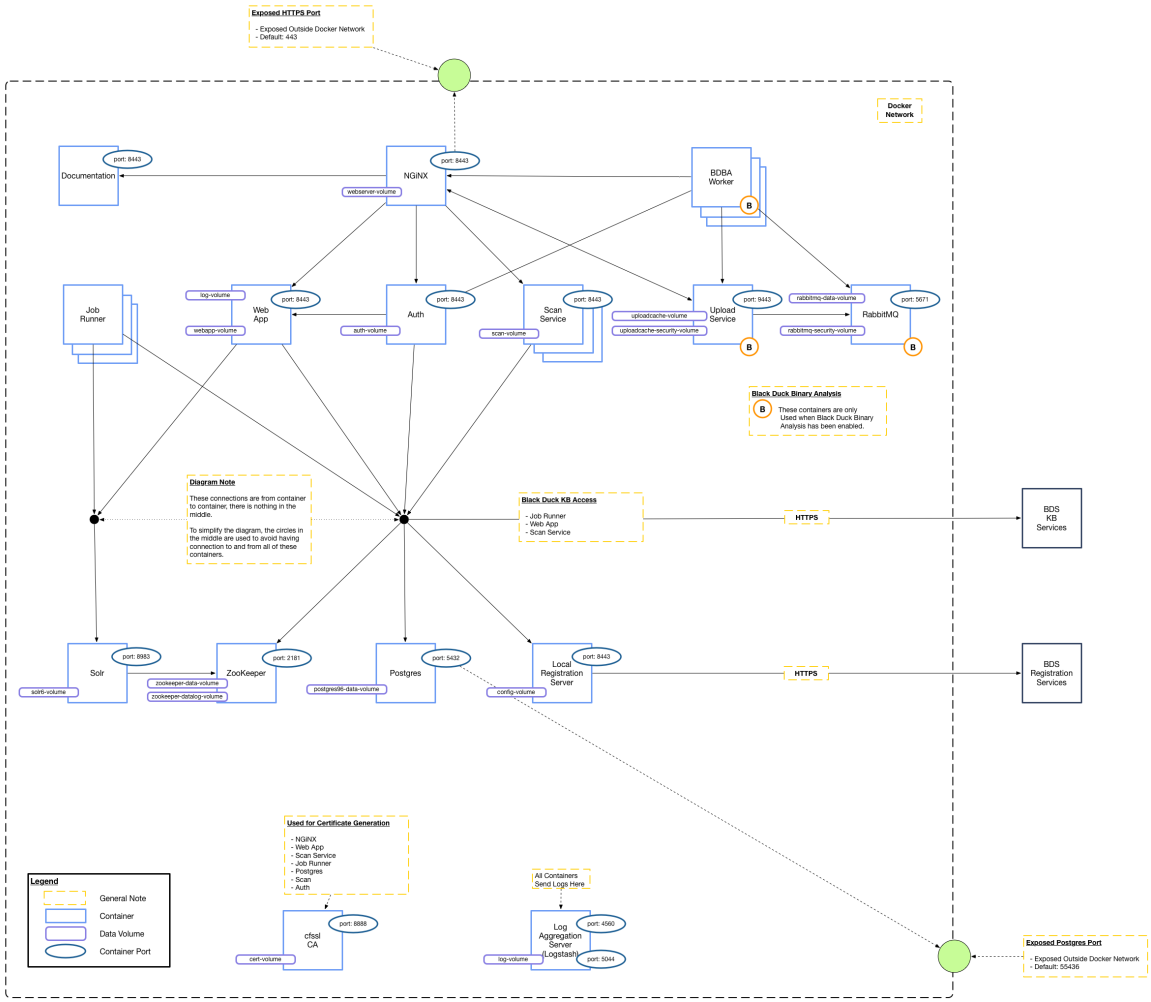
   Note: This container is not included in the Black Duck application if you use an external Postgres instance.

4. Documentation
5. Jobrunner
6. Logstash
7. Registration
8. Scan
9. Solr
10. Webapp
11. WebServer
12. Zookeeper

If Black Duck - Binary Analysis is enabled, there are these additional containers:

1. Binaryscanner
2. RabbitMQ
3. Uploadcache

The following diagram shows the basic relationships among the containers and which ports are exposed outside of the Docker network.



The following tables provide more information on each container.

Authentication container

Container Name: Authentication	
Image Name	blackducksoftware/blackduck-authentication:2019.2.2
Description	The authentication service is the container that all authentication-related requests are made against.
Scalability	There should only be a single instance of this container. It currently cannot be scaled.

Container Name: Authentication	
Links/Ports	Nothing external (8443 internally). This container will need to connect to these other containers/services: • postgres • cfssl • logstash • registration • zookeeper • webapp The container needs to expose 8443 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • postgres - $HUB_POSTGRES_HOST • cfssl - $HUB_CFSSL_HOST • logstash - $HUB_LOGSTASH_HOST • registration - $HUB_REGISTRATION_HOST • zookeeper - $HUB_ZOOKEEPER_HOST • webapp - $HUB_WEBAPP_HOST
Resources/Constraints	• Default max Java heap size: 512MB • Container memory: 1GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

CA container

Container Name: CA	
Image Name	blackducksoftware/blackduck-cfssl:1.0.0
Description	This container uses CFSSL which is used for certificate generation for PostgreSQL, NGiNX, and clients that need to authenticate to Postgres. This container is also used to generate TLS certificates for the internal containers that make up the application.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	The container needs to expose port 8888 within the Docker network to other containers/services that link to it.
Resources/Constraints	• Default max Java heap size: N/A • Container memory: 512MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).
Volumes	cert-volume:/etc/cfssl

DB container

Note: This container is not included in the Black Duck application if you use an external Postgres instance.

Container Name: DB	
Image Name	blackducksoftware/blackduck-postgres:1.0.1
Description	The DB container holds the PostgreSQL database which is an open source object-relational database system. The application uses the PostgreSQL database to store data. There is a single instance of this container. This is where all of the application's data is stored. There are two sets of ports for Postgres. One port will be exposed to containers within the Docker network. This is the connection that the application will use. This port is secured via certificate authentication. A second port is exposed outside of the Docker network. This allows a read-only user to connect via a password set using the `hub_reportdb_changepassword.sh` script. This port and user can be used for reporting and data extraction. Refer to the *Report Database* guide for more information on the report database.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	The DB container needs to connect to these containers/services: • logstash • cfssl The container needs to expose port 5432 to other containers that will link to it within the Docker network. This container exposes port 55436 outside of the Docker network for database reporting.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • logstash: $HUB_LOGSTASH_HOST • cfssl: $HUB_CFSSL_HOST

Container Name: DB	
Resource/Constraints	• Default max Java heap size: N/A • Container memory: 3GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 70. If the container is started as UID 0 (root) then the user will be switched to UID 70:root before executing its main process. This container is not able to start with any other user id.

Documentation container

Container Name: Documentation	
Image Name	blackducksoftware/blackduck-documentation:2019.2.2
Description	The Documentation container supplies documentation for the application.
Scalability	There is a single instance of this container. It should not be scaled.
Links/Ports	This container must connect to these other containers/services: • logstash • cfssl The documentation container must expose port 8443 to other containers that link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • logstash: $HUB_LOGSTASH_HOST • cfssl: $HUB_CFSSL_HOST
Resources/Constraints	▪ Default Max Java Heap Size: 512MB ▪ Container Memory: 512MB ▪ Container CPU: unspecified
Users/Groups	This container runs as UID 8080. If the container is started as UID 0 (root) then the user will be switched to UID 8080:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Jobrunner container

Container Name: Jobrunner	
Image Name	blackducksoftware/blackduck-jobrunner:2019.2.2
Description	The Job Runner container is the container that is responsible for running all of the application's jobs. This includes matching, BOM building, reports, data updates, and so on. This container does not have any exposed ports.
Scalability	This container can be scaled.
Links/Ports	The Job Runner container needs to connect to these containers/services: • postgres • solr • zookeeper • registration • logstash • cfssl
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that any individual service name may be different. For example, you may have an external PostgreSQL endpoint which is resolved through a different service name. To support such use cases, these environment variables can be set to override the default host names: • postgres: $HUB_POSTGRES_HOST • solr: This should be taken care of by ZooKeeper. • zookeeper: $HUB_ZOOKEEPER_HOST • registration: $HUB_REGISTRATION_HOST • logstash: $HUB_LOGSTASH_HOST • cfssl: $HUB_CFSSL_HOST
Resources/Constraints	• Default max Java heap size: 4GB • Container memory: 4.5GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Logstash container

Container Name: Logstash	
Image Name	blackducksoftware/blackduck-logstash:1.0.2
Description	The Logstash container collects and store logs for all containers.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	The container needs to expose port 5044 within the Docker network to other containers/services that will link to it.
Resources/Constraints	• Default max Java heap size: 1GB • Container memory: 1GB • Container CPU: Unspecified
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Registration container

Container Name: Registration	
Image Name	blackducksoftware/blackduck-registration:2019.2.2
Description	The container is a small service that handles registration requests from the other containers. At periodic intervals, this container connects to the Black Duck Registration Service and obtains registration updates.
Scalability	The container should not be scaled.
Links/Ports	The Registration container needs to connect to this containers/services: • logstash • cfssl The container needs to expose port 8443 to other containers that link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment

Container Name: Registration	
	variables can be set to override the default host names: • logstash: $HUB_LOGSTASH_HOST • cfssl: $HUB_CFSSL_HOST
Resources/Constraints	• Default max Java heap size: 512MB • Container memory: 640MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 8080. If the container is started as UID 0 (root) then the user will be switched to UID 8080:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Scan container

Container Name: Scan	
Image Name	blackducksoftware/blackduck-scan:2019.2.2
Description	The scan service is the container that all scan data requests are made against.
Scalability	This container can be scaled.
Links/Ports	This container needs to connect to these containers/services: • postgres • zookeeper • registration • logstash • cfssl The container needs to expose port 8443 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • postgres: $HUB_POSTGRES_HOST • zookeeper: $HUB_ZOOKEEPER_HOST • registration: $HUB_REGISTRATION_HOST • logstash: $HUB_LOGSTASH_HOST

Container Name: Scan	
	• cfssl: $HUB_CFSSL_HOST
Resources/Constraints	• Default max Java heap size: 2GB • Container memory: 2.5GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 8080. If the container is started as UID 0 (root) then the user will be switched to UID 8080:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Solr container

Container Name: Solr	
Image Name	blackducksoftware/blackduck-solr:1.0.0
Description	Solr is an open source enterprise search platform. Black Duck uses Solr as its search server for project data. This container has Apache Solr running within it. There is only a single instance of this container. The Solr container exposes ports internally to the Docker network, but not outside of the Docker network.
Scalability	This container should not be scaled.
Links/Ports	The Solr container needs to connect to these containers/services: • zookeeper • logstash The container needs to expose port 8983 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • zookeeper: $HUB_ZOOKEEPER_HOST • logstash: $HUB_LOGSTASH_HOST

Container Name: Solr	
Resources/Constraints	• Default max Java heap size: 512MB • Container memory: 640MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 8983. If the container is started as UID 0 (root) then the user will be switched to UID 8983:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Webapp container

Container Name: Webapp	
Image Name	blackducksoftware/blackduck-webapp:2019.2.2
Description	The webapp container is the container that all Web/UI/API requests are made against. It also processes any UI requests. In the diagram, the ports for the webapp are not exposed outside of the Docker network. There is an NGiNX reverse proxy (as described in the WebServer container) that is exposed outside of the Docker network instead.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	The webapp container needs to connect to these containers/services: • postgres • solr • zookeeper • registration • logstash • cfssl The container needs to expose port 8443 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • postgres: $HUB_POSTGRES_HOST

Container Name: Webapp	
	• solr: This should be taken care of by ZooKeeper. • zookeeper: $HUB_ZOOKEEPER_HOST • registration: $HUB_REGISTRATION_HOST • logstash: $HUB_LOGSTASH_HOST • cfssl: $HUB_CFSSL_HOST
Resources/Constraints	• Default max Java heap size: 2GB • Container memory: 2.5GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 8080. If the container is started as UID 0 (root) then the user will be switched to UID 8080:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

WebServer container

Container Name: Webserver	
Image Name	blackducksoftware/blackduck-nginx:1.0.2
Description	The WebServer container is a reverse proxy for containers with the application. It has a port exposed outside of the Docker network. This is the container configured for HTTPS. There are config volumes here to allow for the configuration of HTTPS.
Scalability	The container should not be scaled.
Links/Ports	The Web App container needs to connect to these containers/services: • webapp • cfssl • documentation • scan • authentication • upload cache (if Black Duck - Binary Analysis is enabled) This container exposes port 443 outside of the Docker network.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker

Container Name: Webserver	
	Compose or Docker Swarm use. These environment variables can be set to override the default host names: • webapp: $HUB_WEBAPP_HOST • cfssl: $HUB_CFSSL_HOST • scan: $HUB_SCAN_HOST • documentation: $HUB_DOC_HOST • authentication: $HUB_AUTHENTICATION_HOST • upload cache: $HUB_UPLOAD_CACHE_HOST
Resources/Constraints	• Default max Java heap size: N/A • Container memory: 512MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

ZooKeeper container

Container Name: Zookeeper	
Image Name	blackducksoftware/blackduck-zookeeper:1.0.0
Description	This container stores data for the other containers. It exposes ports within the Docker network, but not outside the Docker network.
Scalability	This container should not be scaled.
Links/Ports	The Zookeeper container needs to connect to this container/service: • logstash The container needs to expose port 2181 within the Docker network to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • logstash: $HUB_LOGSTASH_HOST

Container Name: Zookeeper	
Constraints	• Default max Java heap size: 256MB • Container memory: 384MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 1000. If the container is started as UID 0 (root) then the user will be switched to UID 1000:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Black Duck - Binary Analysis containers

The following containers will only be installed if you have Black Duck - Binary Analysis

Binaryscanner container

Container Name: binaryscanner	
Image Name	blackducksoftware/appcheck-worker:1.0.1
Description	This container analyzes binary files. This container is currently only used if Black Duck - Binary Analysis is enabled.
Scalability	This container can be scaled.
Links/Ports	This container needs to connect to these containers/services: • cfssl • logstash • rabbitmq • webserver The container will need to expose port 5671 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • cfssl: $HUB_CFSSL_HOST • logstash: $HUB_LOGSTASH_HOST • rabbitmq: $RABBIT_MQ_HOST • webserver: $HUB_WEBSERVER_HOST

Container Name: binaryscanner	
Resources/Constraints	• Default max Java heap size: N/A • Container memory: 2GB • Container CPU: 1 CPU
Users/Groups	This container runs as UID 0.

Rabbitmq container

Container Name: rabbitmq	
Image Name	blackducksoftware/rabbitmq:1.0.0
Description	This container facilitates upload information to the binary analysis worker. It exposes ports within the Docker network, but not outside the Docker network. This container is currently only used if Black Duck - Binary Analysis is enabled.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	This container needs to connect to these other containers/services: • cfssl The container needs to expose port 5671 to other containers that will link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • cfssl: $HUB_CFSSL_HOST
Constraints	• Default max Java heap size: N/A • Container memory: 1GB • Container CPU: Unspecified
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing it's main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).

Uploadcache container

Container Name: Uploadcache	
Image Name	blackducksoftware/blackduck-upload-cache:1.0.3
Description	This container will be used to temporarily store uploads for binary analysis. It exposes ports within the Docker network, but not outside the Docker network. This container is currently only used if Black Duck - Binary Analysis is enabled.
Scalability	There should only be a single instance of this container. It should not be scaled.
Links/Ports	This container needs to connect to these containers/services: • cfssl • rabbitmq • logstash The container exposes ports 9443 and 9444 to other containers that link to it.
Alternate Host Name Environment Variables	There are times when running in other types of orchestrations that it is useful to have host names set for these containers that are not the default that Docker Compose or Docker Swarm use. These environment variables can be set to override the default host names: • cfssl: $HUB_CFSSL_HOST • logstash: $HUB_LOGSTASH_HOST • rabbitmq: $RABBIT_MQ_HOST
Constraints	• Default max Java heap size: N/A • Container memory: 512MB • Container CPU: Unspecified
Users/Groups	This container runs as UID 100. If the container is started as UID 0 (root) then the user will be switched to UID 100:root before executing its main process. This container is also able to be started as a random UID as long as it is also started within the root group (GID/fsGroup 0).