BLACKDUCK | Hub

Installing Hub using Kubernetes

Version 4.7.3

This edition of the *Installing Hub using Kubernetes* refers to version 4.7.3 of the Black Duck Hub.

This document created or updated on Wednesday, July 25, 2018.

Please send your comments and suggestions to:

Black Duck Software, Incorporated
800 District Avenue, Suite 201
Burlington, MA 01803-5061 USA

Contents

The Hub documentation

The documentation for the Hub consists of online help and these documents:

Title	File	Description
Release Notes	release_notes_bd_hub.pdf	Contains information about the new and improved features, resolved issues, and known issues in the current and previous releases.
Installing Hub using Docker Compose	hub_install_compose.pdf	Contains information about installing and upgrading the Hub using Docker Compose.
Installing Hub using Docker Swarm	hub_install_swarm.pdf	Contains information about installing and upgrading the Hub using Docker Swarm.
Installing Hub using Kubernetes	hub_install_kubernetes.pdf	Contains information about installing and upgrading the Hub using Kubernetes.
Installing Hub using OpenShift	hub_install_openshift.pdf	Contains information about installing and upgrading the Hub using OpenShift.
Getting Started	hub_getting_started.pdf	Provides first-time users with information on using the Hub.
Scanning Best Practices	hub_scanning_best_practices.pdf	Provides best practices for scanning.
Getting Started with the Hub SDK	getting_started_hub_sdk.pdf	Contains overview information and a sample use case.
Report Database	report_db_bd_hub.pdf	Contains information on using the report database.

Hub integration documentation can be found on Confluence.

Training

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

New videos and courses are added monthly.

At Black Duck Academy, 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://www.blackducksoftware.com/services/training

View the full catalog of courses and try some free courses at https://academy.blackducksoftware.com

When you are ready to learn, log in or sign up for an account: https://academy.blackducksoftware.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 Hub using Kubernetes.

Hub Architecture

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

This architecture brings these significant improvements to the Hub 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 Hub 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 the Black Duck Hub:

- Registration server: Used to validate the Hub 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 the Hub can display the most up-to-date information about open source software (OSS) without requiring regular updates to your Hub.

Chapter 2: Installation planning

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

Getting started

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

New installations

For new installation of the Hub:

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 the Hub

1. Read this planning chapter to review all requirements,
2. After ensuring that you meet all requirements, go to Chapter 6 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 Hub containers
- Commensurate space for database backups

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 and Scan 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 the Hub, ensure that your Kubernetes system administrator has put enterprise-level metrics and logging in place to identify unhealthy nodes on the cluster.

Kubernetes requirements

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

There are two restrictions when using Hub in Kubernetes:

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

  Storage must be provided for this node.

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

- The hub-webapp service and the hub-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.

Operating systems

The Dockerized Hub 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

The Hub 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 the Hub:

- Chrome 66.0.3359.181 (Official Build) (64-bit)
- Firefox 60.0.1 (64-bit)
- Internet Explorer 11.0.1029.15063.0
- Microsoft Edge 40.15063.674.0

- Microsoft EdgeHTML 15.15063
- Safari 11.1 (13605.1.33.1.4)

Note that the Hub does not support compatibility mode.

Note: These browser versions are the currently-released versions on which Black Duck has tested Hub. Newer browser versions may be available after the Hub 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

The Hub requires the following ports to be externally accessible:

- Port 443 – Web server HTTPS port for the Hub 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 Hub 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

The Hub uses the PostgreSQL object-relational database to store data.

For the Hub version 4.7.3, you must use PostgreSQL version 9.6.x for compatibility with the Hub version 4.7.3. Refer to Upgrading the Hub for database migration instructions if upgrading from a pre-4.2.0 version of the Hub.

Prior to installing the Hub, 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 Hub 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

The Hub supports:

- No Authentication
- Digest
- Basic
- NTLM

If you are going to make proxy requests to the Hub, 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 the Hub

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 the Hub 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 the Hub.

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

  Currently the Hub requires PostgreSQL version 9.6.x.

Chapter 3: Installing the Hub

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

Hub 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 Hub 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 Hub `.gz` file:

   ```
   gunzip hub-4.7.3.tar.gz
   ```

3. Unpack the Hub `.tar` file:

   ```
   tar xvf hub-4.7.3.tar
   ```

Download using the wget command

1. Run the following command:

   ```
   wget https://github.com/blackducksoftware/hub/archive/v4.7.3.tar.gz
   ```

2. Uncompress the Hub `.gz` file:

   ```
   gunzip v4.7.3.tar.gz
   ```

3. Unpack the Hub `.tar` file:

   ```
   tar xvf v4.7.3.tar
   ```

Distributions

The following is a list of files in the distribution:

- `external-postgres-init.pgsql`
- `3-hub.yml`
- `2-postgres-db-internal.yml`
- `2-postgres-db-external.yml`
- `1-cm-hub.yml`
- `1-cfssl.yml`

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 the Hub.

Using persistent volumes

Both the Hub's PostgreSQL database and various Hub components have data that must be stored. The

default configuration files in the Hub installation media specify using `emptyDir` for storage, which provides temporary storage of data. This minimizes complexity, but could result in data loss if Hub containers are restarted and rescheduled.

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

Setting up persistent volumes

Before the Hub 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 the Hub configuration files to create persistent volume claims against the volumes. Do not proceed with a production Hub installation until persistent volumes are available in your cluster.

Creating a namespace

Create a virtual cluster, or namespace, for running the Hub 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 the Hub, 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.

Customizing your Hub configuration files

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

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 Hub installation, then it is highly recommended that you configure your Hub to leverage persistent volume claims for persistent storage. To do so, refer to the Adding a persistent volume claim to a Hub 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	pv-bd-hub-cfssl

Installing a PostgreSQL database inside a container within the cluster

The Black Duck Hub requires a PostgreSQL 9.6 database for all the Hub'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 Hub installation, you must configure the database to use a persistent volume claim. To do so, refer to Adding a persistent volume claim to a Hub 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	pv-bd-hub-postgres

General Hub configuration

1. If this is a production Hub installation, then Black Duck highly recommends that you configure your Hub to leverage persistent volume claims for persistent storage. If this is a non-production Hub, this step can be skipped. There are several Hub services that must be configured. To do so, refer to Adding a persistent volume claim to a Hub 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
webserver	pvclaim-bd-hub-webserver	1Gi	dir-webserver
registration	pvclaim-bd-hub-registration	100Mi	dir-registration
zookeeper	pvclaim-bd-hub-zookeeper	1Gi	dir-zookeeper
scan	pvclaim-bd-hub-scan	100Mi	dir-scan
authentication	pvclaim-bd-hub-authentication	100Mi	dir-authentication

2. If you are using a network proxy:

   a. These Hub 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 the Hub

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

Before you begin

Black Duck 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
```

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 the Hub. 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 the Hub

Now that PostgreSQL is installed, it must be initialized with Hub-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 the Hub.

Note: To use the reporting database in the Hub, 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

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>'"
```

Creating Hub containers

Now that your PostgreSQL database is configured, you can create the Hub 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 the Hub is installed.

Starting Over

If you need to edit the `3-hub.yml` file and re-create the Hub, 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 Hub installation in Kubernetes

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

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

Connecting to the Hub

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

https://hub.example.com

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

Enter the registration key provided to you to access the Hub.

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 Job Runner and Scan 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-Hub server.
- Configuring secure LDAP.
- Configuring Single Sign-On (SSO).
- Enabling the hierarchical BOM.
- Backing up PostgreSQL volumes.

Understanding the default sysadmin user

When you install the Black Duck Hub, 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 Hub UI.

Environment variables

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

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 Hub installation in Kubernetes, this External Load Balancer will forward web traffic to the Hub's NGiNX proxy server, which sends traffic along to the Hub'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 Hub'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, the Hub 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 the Hub 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 Hub server when scanning as the Hub Scanner cannot verify the certificate because it is a self-signed and is not issued by a CA. Note that the Hub Scanner 2.0 does provide an option that allows you to connect to the Hub instance with a self-signed certificate.

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 Hub 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 the Hub 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 Hub's web server.

1. To use a custom certificate, create two Kubernetes secrets called `WEBSERVER_CUSTOM_CERT_FILE` and `WEBSERVER_CUSTOM_KEY_FILE` with the custom certificate and custom key, respectively, in your namespace:

   ```
   kubectl secret create WEBSERVER_CUSTOM_CERT_FILE --from-file=<certificate file>
   kubectl secret create WEBSERVER_CUSTOM_KEY_FILE --from-file=<key file>
   ```

2. In the `3-hub.yml` file, find the commented section 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: {}
  # 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
```

Scaling Job Runner and Scan containers

The Job Runner and Scan 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
```

Configuring the report database password

A PostgreSQL report database provides access to the Hub 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 the Hub, 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 the Hub. 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 Hub 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 Hub path. For example, if you have Hub 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 the Hub, add the environment variable above into the hub-nginx image stanza in the `3-hub.yml` file.

Accessing the REST APIs from a non-Hub server

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

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

The properties used to enable and configure CORS for Hub 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 the Hub, add the environment variables above into the hub-nginx image stanza in the `3-hub.yml` file.

Configuring secure LDAP

If you see certificate issues when connecting your secure LDAP server to the Hub, the most likely reason is that the Hub 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 Hub

LDAP truststore by:

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

Obtaining your LDAP information

Contact your LDAP administrator and gather the following information:

LDAP Server Details

This is the information that the Hub 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 the Hub 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

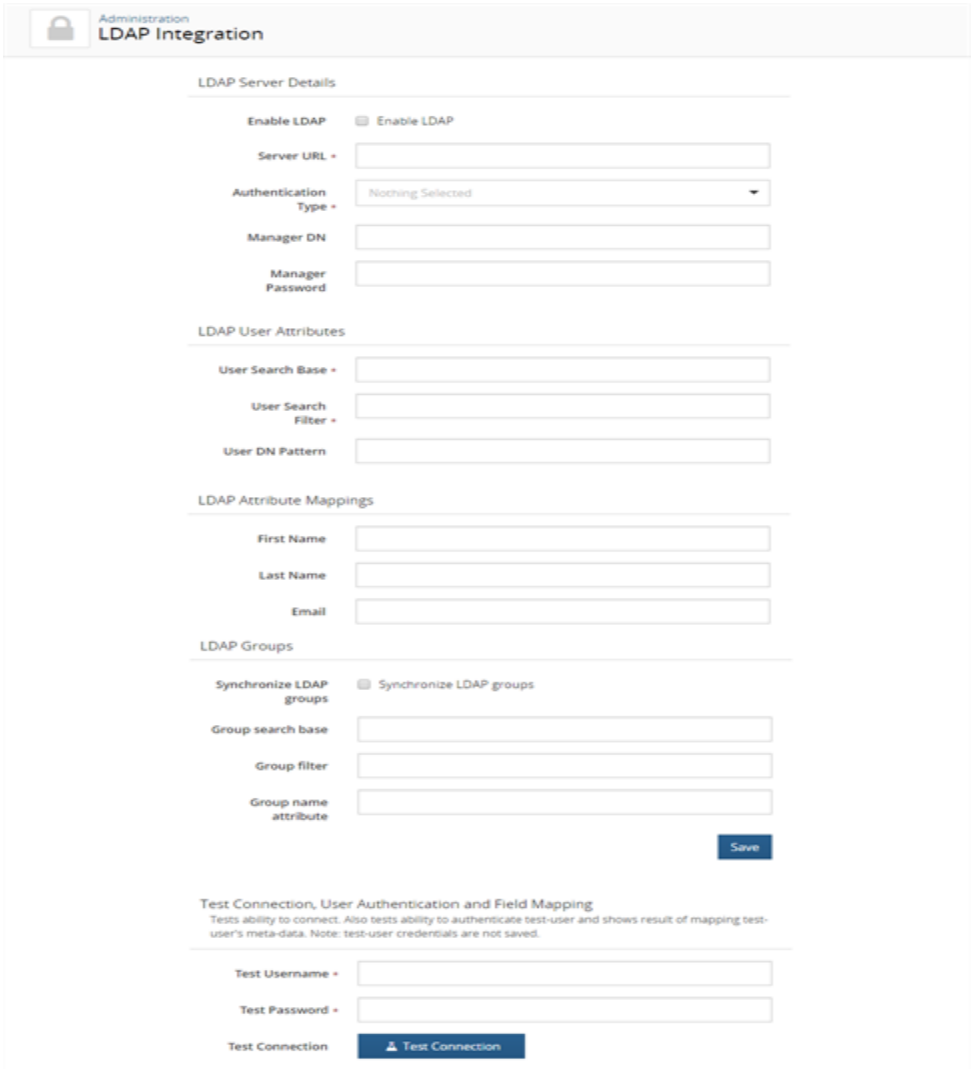
To import the server certificate

1. Log in to the Hub 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.

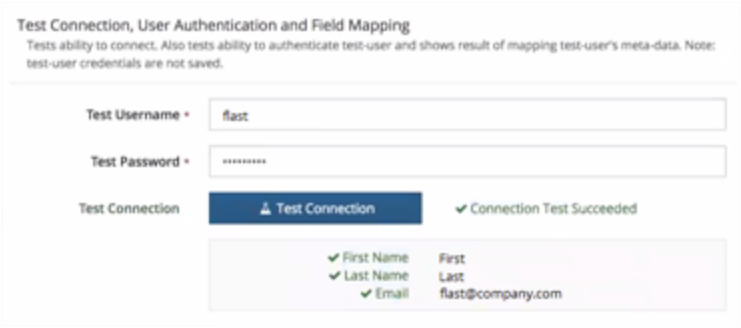


4. Select the **Enable LDAP** option and complete the information in the **LDAP Server Details** and **LDAP User Attributes** sections, as described above. In the **Server URL** field, ensure that you have configured the secure LDAP server: the protocol scheme is ldaps://.

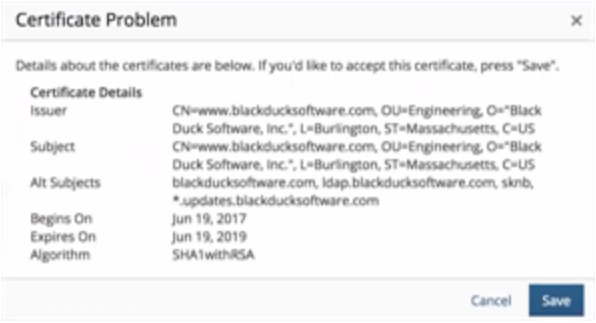
5. Enter the user credentials in the **Test Connection, User Authentication and Field Mapping** section and click **Test Connection**.

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 Hub's SAML implementation provides single sign-on (SSO) functionality, enabling Hub users to be automatically signed-in to the Hub when SAML is enabled. Enabling SAML applies to all your Hub users, and cannot be selectively applied to individual users.

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:

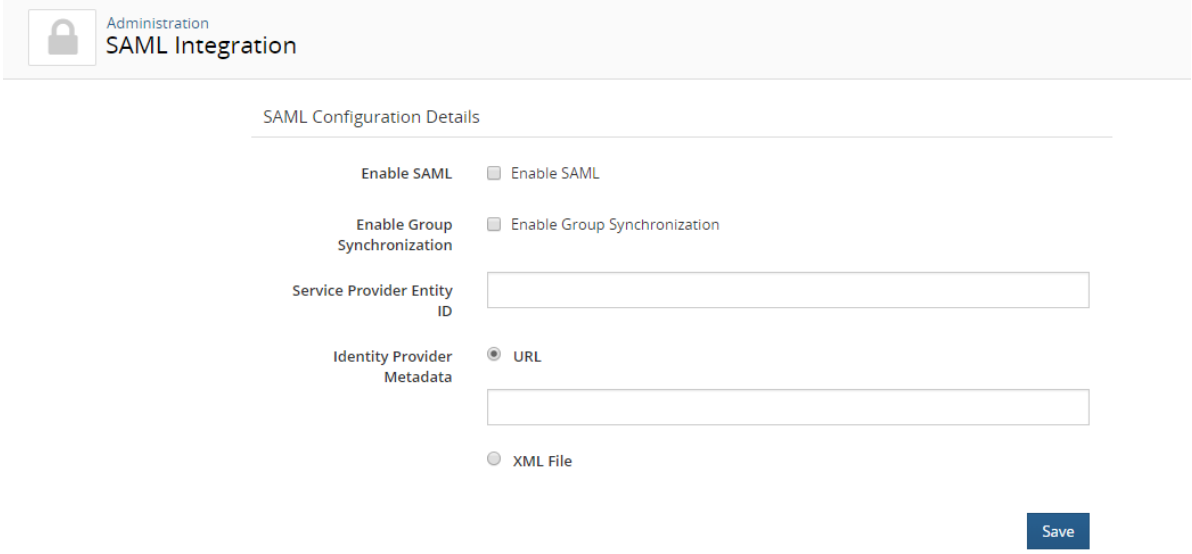
- The Hub is able to synchronize and obtain an external user's information (Name, FirstName, LastName and Email) if the information is provided in attribute statements. Note that the first and last name values are case-sensitive.

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

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 IDP are created in the Hub 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. **Service Provider Entity ID** field. Enter the information for the Hub server in your environment in the format **https://*host*** where *host* is your Hub server.
   d. **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.
4. Click **Save**.
5. Add the HUB_SAML_EXTERNAL_URL to your `hub-proxy.env` file (for Docker Swarm or Docker Compose) or the `3-hub.yml` file (for Kubernetes or OpenShift). The value is the public URL of the Hub server. For example:

   ```
   HUB_SAML_EXTERNAL_URL=https://blackduck-docker01.dc1.lan
   ```

Note: You must restart the Hub for your configuration changes to take effect.

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**.

Note: You must restart the Hub for your configuration changes to take effect.

Enabling the hierarchical BOM

By default, the hierarchical BOM is disabled. To enable this feature, add the HIERARCHICAL_VERSION_BOM environment variable to the `1-cm-hub.yml` file located in the `kubernetes` directory. Set the value to "true".

Resetting the value to "false" disables the feature.

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.

Chapter 5: Upgrading the Hub

This chapter describes how to upgrade an existing Hub on Kubernetes to a newer version of the Hub on Kubernetes.

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

Upgrading the Hub on Kubernetes

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

Backing up the PostgreSQL database

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

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

- Backing up AppMgr Hub data (for migration purposes)
- Backing up Hub Kubernetes PostgreSQL data
- Restoring Hub 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 Hub 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 the Hub 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 the Hub server as the **blckdck** user.

   Note: This is the user that owns the Hub 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 Hub database (the one that comes standard with the Kubernetes Hub), you must locate the PostgreSQL node, SSH into it, and run a data-dump script that creates a local backup 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}'
   ```

   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 the command:

   ```
   ./bin/hub_create_data_dump.sh <path to local PostgreSQL dump file>
   ```

3. Run the following script which creates a PostgreSQL dump file in the hub-postgres container and

then copies the dump file from the container to the local PostgreSQL dump file.

```
./bin/hub_create_data_dump.sh <path to local PostgreSQL dump file>
```

Important: You must run the `hub_create_data_dump.sh` script *before* upgrading the Hub using the version of the script located in the pre-upgrade directory.

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 the command:

   ```
   ./bin/hub_db_migrate.sh <path to dump file>
   ```

Error messages

When the dump file is restored from the an AppMgr installation of the Hub, 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 Hub

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

The command to upgrade a container in Kubernetes is:

```
kubectl set image <image> hub-image=hub-image:version
```

The following Hub containers each needs to be individually updated:

- hub-cfssl
- hub-documentation

- hub-postgres
- hub-jobrunner
- hub-webapp
- hub-webserver
- hub-logstash
- hub-registration
- hub-solr
- hub-zookeeper
- hub-scan
- hub-authentication

For example, here are the specific commands that must be run to upgrade to the Hub 4.7.3:

```
kubectl set image deployment/cfssl cfssl=cfssl:4.7.3

kubectl set image deployment/documentation
documentation=documentation:4.7.3

kubectl set image deployment/jobrunner jobrunner=jobrunner:4.7.3

kubectl set image deployment/postgres postgres=postgres:4.7.3

kubectl set image deployment/webapp-logstash webapp=webapp:4.7.3

kubectl set image deployment/webapp-logstash logstash=logstash:4.7.3

kubectl set image deployment/webserver-logstash webserver=webserver:4.7.3

kubectl set image deployment/registration registration=registration:4.7.3

kubectl set image deployment/solr solr=solr:4.7.3

kubectl set image deployment/zookeeper zookeeper=zookeeper:4.7.3

kubectl set image deployment/scan scan=scan:4.7.3

kubectl set image deployment/authentication
authentication=authentication:4.7.3
```

Appendix A: Adding a persistent volume claim to a Hub service

This appendix describes how to request persistent storage for a particular Hub 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 Hub service expects a particular mount path and is it already correctly specified.

3. Save and close the configuration file.

Appendix B: Troubleshooting

If a particular Hub 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 Hub UI

1. Log in to the Hub 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.

Appendix C: Containers

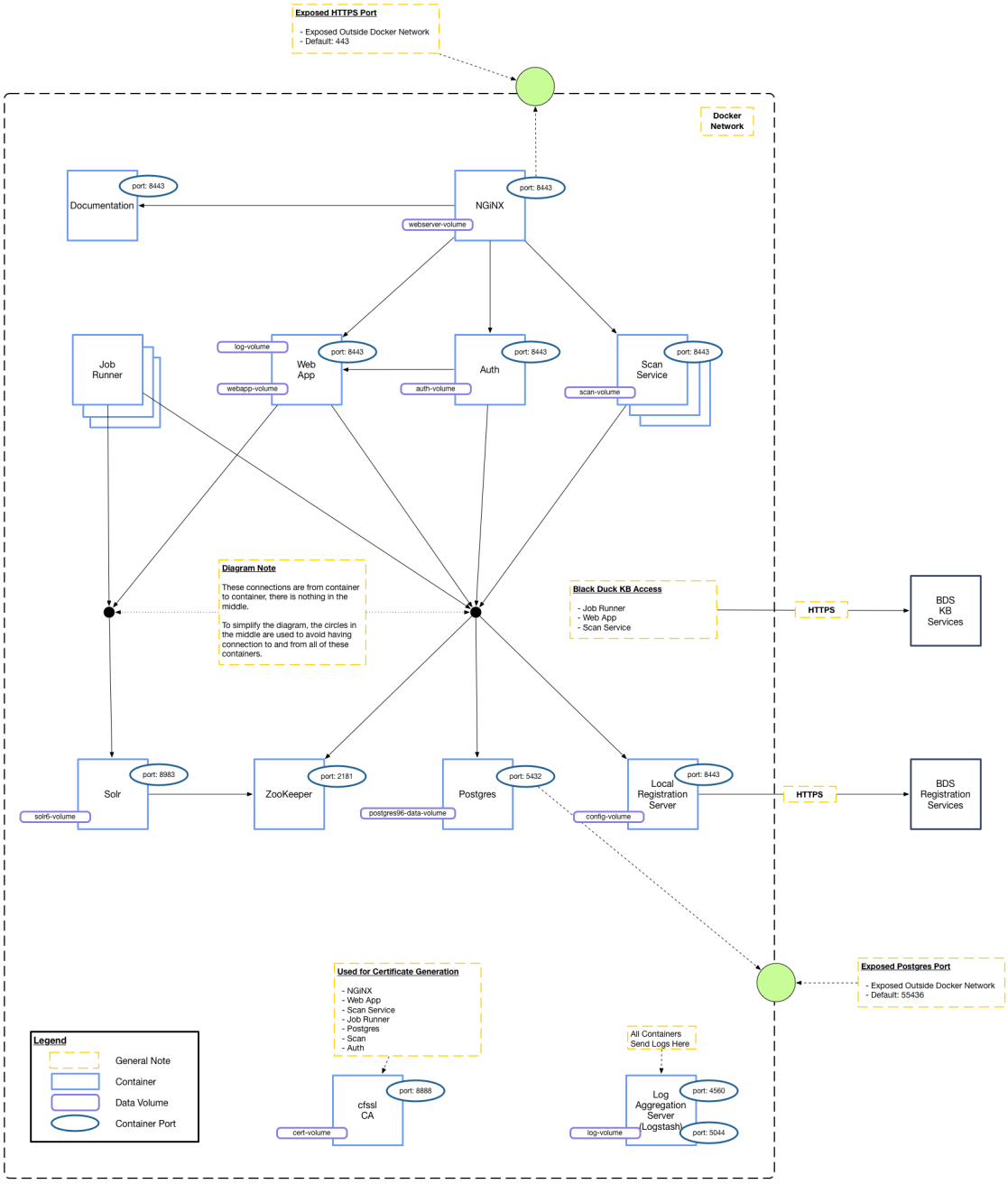
These are the containers within the Docker network that comprise the Hub application:

1. Web App
2. Authentication
3. Scan
4. Job Runner
5. Solr
6. Registration
7. DB

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

8. Documentation
9. WebServer
10. Zookeeper
11. LogStash
12. CA

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.

Web App container

Container Name: Web App	
Image Name	blackducksoftware/hub-webapp:4.7.3
Description	The Web App 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 Web App 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 Web App 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 • 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
Constraints	• Default max Java heap size: 2GB • Container memory: 2.5GB • Container CPU: 1 CPU
Volumes	log-volume:/opt/blackduck/hub/logs webapp-volume:/opt/blackduck/hub/hub-webapp/security

Authentication container

Container Name: hub-authentication	
Image Name	blackducksoftware/hub-authentication:4.7.3
Description	The Hub 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.
Links/Ports	Nothing external (8443 internally). This container will need to connect to these other containers/services: • postgres • cfssl • logstash • regtistration • 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
Constraints	• Default max Java heap size: 512MB • Container memory: 1GB • Container CPU: 1 CPU
Volumes	authentication-volume: /opt/blackduck/hub/hub-authentication/security

Scan container

Container Name: Scan	
Image Name	blackducksoftware/hub-scan:4.7.3
Description	The Hub scan service is the container that all scan data requests are made against.
Scalability	This container can be scaled.
Links/Ports	The Scan 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 • cfssl: $HUB_CFSSL_HOST
Constraints	• Default max Java heap size: 2GB • Container memory: 2.5GB • Container CPU: 1 CPU
Volumes	log-volume:/opt/blackduck/hub/logs scan-volume:/opt/blackduck/hub/hub-scan/security

Job runner container

Container Name: Job Runner	
Image Name	blackducksoftware/hub-jobrunner:4.7.3
Description	The Job Runner container is the container that is responsible for running all Hub 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
Constraints	• Default max Java heap size: 4GB • Container memory: 4GB • Container CPU: 1 CPU
Volumes	N/A

Solr container

Container Name: Solr	
Image Name	blackducksoftware/hub-solr:4.7.3
Description	Solr is an open source enterprise search platform. The Hub 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 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: • zookeeper: $HUB_ZOOKEEPER_HOST • logstash: $HUB_LOGSTASH_HOST
Constraints	• Default max Java heap size: 512MB • Container memory: 512MB • Container CPU: Unspecified
Volumes	solr6-volume:/opt/blackduck/hub/solr/cores.data

Registration container

Container Name: Registration	
Image Name	blackducksoftware/hub-registration:4.7.3
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.

Container Name: Registration	
Links/Ports	The Registration container needs to connect to this containers/services: • logstash 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 variables can be set to override the default host names: • logstash: $HUB_LOGSTASH_HOST
Constraints	• Default max Java heap size: 256MB • Container memory: 256MB • Container CPU: Unspecified
Volumes	config-volume:/opt/blackduck/hub/registration/config

DB container

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

Container Name: DB	
Image Name	blackducksoftware/hub-postgres:4.7.3
Description	The DB container holds the PostgreSQL database which is an open source object-relational database system. The Hub uses the PostgreSQL database to store data. There is a single instance of this container. This is where all Hub 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 Hub App, Job Runner, and potentially other containers 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.

Container Name: DB	
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
Constraints	• Default max Java heap size: N/A • Container memory: 3GB • Container CPU: 1 CPU
Volumes	postgres96-data-volume:/var/lib/postgresql/data

Documentation container

Container Name: Documentation	
Image Name	blackducksoftware/hub-documentation:4.7.3
Description	The Documentation container supplies documentation for the Hub.
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 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:

Container Name: Documentation	
	• logstash: $HUB_LOGSTASH_HOST
Constraints	▪ Default Max Java Heap Size: 512MB ▪ Container Memory: 512MB ▪ Container CPU: unspecified
Volumes	N/A

WebServer container

Container Name: WebServer	
Image Name	blackducksoftware/hub-nginx:4.7.3
Description	The WebServer container is a reverse proxy for the Hub Web App. It has a port exposed outside of the Docker network. This is the container configured for HTTPS. There are config volumes here for 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 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 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

Container Name: WebServer	
Constraints	• Default max Java heap size: N/A • Container memory: 512MB • Container CPU: Unspecified
Volumes	webserver-volume:/opt/blackduck/hub/webserver/security

ZooKeeper container

Container Name: Zookeeper	
Image Name	blackducksoftware/hub-zookeeper:4.7.3
Description	This container stores data for the Hub App, Job Runners, Solr, and potentially 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
Constraints	• Default max Java heap size: 256MB • Container memory: 256MB • Container CPU: Unspecified
Volumes	N/A

LogStash container

Container Name: LogStash	
Image Name	blackducksoftware/hub-logstash:4.7.3
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.

Container Name: LogStash	
Links/Ports	The container needs to expose port 5044 within the Docker network to other containers/services that will link to it.
Constraints	• Default max Java heap size: 1GB • Container memory: 1GB • Container CPU: Unspecified
Volumes	log-volume:/var/lib/logstash/data

CA container

Container Name: CA	
Image Name	blackducksoftware/hub-cfssl:4.7.3
Description	The CA container uses CFSSL which is used for certificate generation for PostgreSQL, NGiNX, and clients that need to authenticate to Postgres.
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.
Constraints	• Default max Java heap size: N/A • Container memory: 512MB • Container CPU: Unspecified
Volumes	cert-volume:/etc/cfssl