BLACKDUCK | Hub

Installing Hub using Kubernetes

Version 4.3.1

This edition of the *Installing Hub using Kubernetes* refers to version 4.3.1 of the Black Duck Hub.

This document created or updated on Friday, November 10, 2017.

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:

- 4 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 container).

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 1.6. and 1.7, 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 62.0.3202.62 (Official Build) (64-bit)
- Firefox 56.0.0
- Internet Explorer 11.0.43 (KB4032782)
- Microsoft Edge 40.15063.0.0

- Microsoft EdgeHTML 15.15063
- Safari 11.0 (12604.1.38.1.7)

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. Network proxy information can be expressed as environment variables and placed in the Hub's pod.env file. See the pod.env and the other.env files in GitHub for more information.

Database requirements

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

Prior to installing the Hub, determine whether you want to use the database container that is automatically installed or an external (for example, Amazon Relational Database Service (RDS)) PostgreSQL instance.

To use an external PostgreSQL instance:

1. Set up your external PostgreSQL instance using Amazon RDS.

   When creating your RDS instance, set the "Master User" to **blackduck**.

2. Configure your database connection settings.
3. Install or upgrade the Hub.

Currently, the Hub requires PostgreSQL 9.6.X.

PostgreSQL versions

For the Hub version 4.3.1, the currently-supported version of PostgreSQL is 9.6.x, which is the version

supplied in the Hub's PostgreSQL container. If you choose to run your own PostgreSQL instance, you must be at PostgreSQL version 9.6.x for compatibility with the Hub version 4.3.1.

Refer to Chapter 6, Upgrading the Hub for database migration instructions if upgrading from a pre-4.2.0 version of the Hub.

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.

Click here for more information on configuring an external PostgreSQL server.

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. Specifically, you have configured database settings if you are using an external PostgreSQL instance.
Proxy requirements	
☐	You have ensured that your network meets the proxy requirements. Configure proxy settings before or after installing the Hub.
Web server requirements	
☐	Configure web server settings before or after installing the Hub.

1. 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.

```
wget https://github.com/blackducksoftware/hub/archive/v4.3.1.tar.gz
```

As part of the install/upgrade process, these orchestration files pull down the necessary Docker images.

Uncompress the Hub `.gz` file:

```
gunzip hub-4.3.1.tar.gz
```

Unpack the Hub `.tar` file:

```
tar xvf hub-4.3.1.tar
```

Distributions

The following is a list of files in the distribution:

- `external-postgres-init.pgsql`: PostgresSQL.sql file used to configure an external PostgreSQL database.
- `kubernetes-external-rds.yml`: Creates Kubernetes deployments which operate against an external database.
- `kubernetes-post-db.yml`: Creates Kubernetes deployments to run after a database migration.
- `kubernetes-pre-db.yml`: Creates Kubernetes deployments to run as part of database migration or for bootstrapping.
- `other.env`: Defines additional, optional environment variables for containers that run in the pod.
- `pods.env`: Defines environment variables for the containers that run in the pod.

From the bin directory in the distribution:

- `hub_create_data_dump.sh`: Script used to back up the PostgreSQL database when using the database container provided by the Hub.
- `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.

2. 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.

3. Creating the config map

There are several environment variable settings that can be used with Kubernetes. You can upload these environment variables as a configmap for the Hub.

Throughout this installation guide, references are made to a pods.env file, which stores all Hub configuration data. This method was used to consolidate all information into one resource to simplify managing watches and configuration-related logic. The file can be separated if you wanted to hide environment variables from different pods.

In the example below, the config map is created using the variables in pods.env, plus a namespace variable specified directly on the command line:

```
kubectl create -f pods.env --namespace=my-ns
```

Note: The other.env file has a list of other environment variables, which you can optionally copy into the pods.env file when you create your config map.

See Chapter 4, Administrative Tasks, for more information on environment variables that can be configured.

4. Installing the Hub containers

There are four steps to install the Hub services in Kubernetes:

a. Identify and label the PostgreSQL node and create the data store.

b. Install the PostgreSQL/cfssl containers so they can be available for the other Hub services.

c. Migrate any data from a previous/legacy non-Kubernetes Hub.

d. Install/run the remaining Hub containers.

a. Identify and label the PostgreSQL node and create the data store

Note: Skip this step if you plan to use an external PostgreSQL.

Identifying and labeling the PostgreSQL node

1. Determine which node will run PostgreSQL. The name of the node you choose must be one that can be seen by running the following command:

   ```
   kubectl get nodes --namespace=my-ns
   ```

You should see output similar to the following:

```
NAME                                          STATUS    AGE       VERSION
gke-temp-saas-0-default-pool-3dd610fe-141r    Ready     11d       v1.6.9
gke-temp-saas-0-default-pool-3dd610fe-2mr1    Ready     19d       v1.6.9
gke-temp-saas-0-default-pool-3dd610fe-43w5    Ready     11d       v1.6.9
gke-temp-saas-0-default-pool-3dd610fe-5nxm    Ready     19d       v1.6.9
```

2. Once you have determined which node PostgreSQL will run on, label this node as follows:

   ```
   kubectl label nodes <node-name> blackduck.hub.postgres=true --
   namespace=my-ns
   ```

   This label is required for PostgreSQL host-storage. With PostgreSQL host-storage, if PostgreSQL were migrated to a new node, the data directory might not exist.

Creating a data store

Now that you have determined and labeled the node, create a directory on the node where the data will reside. If you have complete control of your cluster, you can SSH into this node, and create a data directory, as follows:

```
mkdir -p /var/lib/hub-postgresql/data && chmod -R 775
/var/lib/hubpostgresql/data
```

Note that the command given above is just one of many ways to satisfy the basic need for a persistent data location in a node that Kubernetes can schedule (assign) pods to.

For example, in a production Kubernetes cluster, you may want to configure volumes differently by changing the hostPath volume definition in the postgres pod. Consult with your Kubernetes administrator, or with Black Duck support, to determine the storage model that works best for your organization's needs.

b. Installing the PostgreSQL/cfssl containers

Note: Skip this step if you plan to use an external PostgreSQL.

If you plan to use the internal Hub PostgreSQL container for data storage, run the following command:

```
kubectl create -f kubernetes-pre-db.yml --namespace=my-ns
```

To verify that the command succeeded, run:

```
kubctl get pods --namespace=my-ns
```

You should see pods corresponds to PostgreSQL and cfssl.

c. Migrating data from a previous version of the Hub

If you intend to migrate data from a previously-existing version of the Hub, follow these instructions for backing up and restoring Hub data. If not, this step can be skipped.

d. Installing/Running the remaining Hub containers

Now that the database has been set up, run one of the following commands to create the full Kubernetes deployment of the Hub:

- With the PostgreSQL database container:

```
kubectl create -f kubernetes-post-db.yml --namespace=my-ns
```

- With an external PostgreSQL instance:

```
kubectl create -f kubernetes-external-rds.yml --namespace=my-ns
```

Verifying the Hub containers are running

After you have created the full Kubernetes deployment of the Hub, confirm that the installation was successful by running the following command to see the running Hub containers:

```
kubctl get pods --namespace=my-ns
```

Note: See Docker containers for the full list of containers in the Kubernetes Hub.

If you suspect that the proper Hub Docker images cannot be pulled from the appropriate repository, debug this issue by looking at Kubernetes events by running the following command:

```
kubectl get events
```

The output of this command could give hints on permissions or networking limitations that may impact pulling images.

5. Exposing your Kubernetes service

Immediately after the installation of the Hub in Kubernetes, the IP addresses of the containers are internal (10.x), and are not visible/routable to the Internet. To make Hub services, such as the Web UI, externally available, expose your Kubernetes service using the following command:

```
kubectl expose --namespace=default deployment nginx-webapp-logstash --
type=LoadBalancer --port=443 --target-port=8443 --name=nginx-gateway
```

The command above leverages your cloud provider's native load balancer implementation to provision an externally addressable IP address for your Black Duck Hub. Specifically, external packets addressed to this IP address at port 443 will be forwarded (via packet forwarding, for example, in GKE) to the Hub's NGiNX container at port 8443 (the NGiNX default).

Alternatively, use the `--type=NodePort` option which lets you access the service at any port. You would use this if you do not have (or do not wish to use) an external load balancer; contact your system administrator for guidance.

To verify that the provisioning of the load balancer (above) worked properly, find its external endpoint by entering the following command:

```
kubectl get services -o wide
```

A response such as the following appears:

```
nginx-gateway 10.99.200.3 a0145b939671d... 443:30475/TCP 2h
```

You can connect to the Hub's NGiNX services with a utility like curl as follows:

```
ubuntu@ip-10-0-22-242:~$ curl --insecure
https://a0145b939671d11e7a6ff12207729cdd-587604034.us-east-
1.elb.amazonaws.com:443
```

And you should be able to see a result which includes an HTTP page.

```
<!DOCTYPE html><html lang="en"><head><meta charset="utf-8"><meta...
```

This resulting HTML is the sign that your Webapp is now reachable.

Note: If you need to make your PostgreSQL container externally accessible (for example, for third-party or database applications running outside of the Kubernetes cluster), you will have to run through an analogous process for the PostgreSQL container.

Connecting to the Hub

Once all of the 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.
- Configuring the Hub session timeout value.
- Configuring an external PostgreSQL instance.
- Replacing the existing self-signed certificate for the Web Server with a custom certificate.
- Accessing log files.
- Scaling job runners.
- 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).
- 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. The other.env file contains these environment variables, which can be set and added to either the pods.env file (specifically the config-map file) or directly to the Hub distribution's .yml files (via name-value pairs). See Creating the configmap on how to invoke the pods.env file when you create a namespace, and see the previous chapter for launching the Kubernetes cluster with the yml files.

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 pods.env 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 pods.env 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 pods.env file to 1.

Proxy settings

There are currently three services requiring access to services hosted by Black Duck Software:

- Registration
- Jobrunner
- Webapp

If a proxy is required for external internet access, you must configure it.

To configure the proxy for external internet access:

1. Create a HUB PROXY SECTION in pods.env file by copying the section from the other.env file.
2. Add the required parameters for your proxy setup.

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.

Authenticated proxy password

There are two methods for specifying a proxy password when using Kubernetes:

- Specify an environment variable called HUB_PROXY_PASSWORD that contains the proxy password.
- Add a Kubernetes secret called HUB_PROXY_PASSWORD_FILE.

Environment Variables

The easiest method is to specify an environment variable called HUB_PROXY_PASSWORD in the pods.env file that contains the proxy password.

Kubernetes Secret

The most secure way to specify the proxy password is to add it as a Kubernetes secret and to inject that secret into the pod.

To store the proxy password as a secret, place the password in a file (called hpp in the following example), then run the kubectl create secret command:

```
echo "mypassword1234" > hpp
kubectl create secret generic hub-proxy-password --from-file=./hpp
```

After running these commands, for safety reasons, delete the hpp file.

Now the that the secret has been created in Kubernetes, you must expose the secret to the Hub services – Webapp, Registration, Jobrunner – that require access to it.

For example, the following text added to the .yml file exposes the secret to Webapp:

In each of these pod specifications, add the secret injection next to the image that is using them.

For example, add the following text to the .yml file you use to launch the cluster to expose the secret to the Webapp service (for example, kubernetes-external-rds.yml or kubernes-post-db.yml):

```
image: hub-webapp:4.3.1
env:
- name: HUB_PROXY_PASSWORD

 valueFrom:
   secretKeyRef:
```

```
    name: hub-proxy-password
    key: hpp
```

Add this section of text for the Registration and Jobrunner services by replacing `image: hub-webapp:4.3.1` with `image: registration:4.3.1` and `image: jobrunner:4.3.1`, respectively.

Configuring the Hub session timeout

By default, the Hub session timeout value is 2 hours.

To specify a different value, use the HUB_WEBAPP_SESSION_TIMEOUT property to specify the new timeout value in number of seconds. For example, a timeout value of one hour would be 3600 seconds.

As with all other environment variables, you can specify the values in the pods.env file, but it is usually better to inject the variable directly into the webapp container by specifying it as a name/value pair in the .yml file you use to launch the cluster (for example, kubernetes-external-rds.yml or kubernes-post-db.yml). For example:

```
image: hub-webapp:4.3.1
env:
- name: HUB_WEBAPP_SESSION_TIMEOUT
  value: 3600
```

Configuring an external PostgreSQL instance

The Hub supports using an external PostgreSQL instance. The external PostgreSQL instance needs to be initialized and information must be provided to the Webapp and Jobrunner containers.

PostgreSQL configuration

1. Create a database user named **blackduck** with administrator privileges.

   For Amazon RDS, set the "Master User" to **blackduck** when creating the database instance.

   No other specific values are required.

2. Run the `external-postgres-init.pgsql` script to create users, databases, and other necessary items. For example:

   ```
   psql -U blackduck -h <hostname> -p <port> -f external_postgres_init.pgsql postgres
   ```

3. Configure passwords for the **blackduck**, **blackduck_user**, and **blackduck_reporter** database users.

   These users were created by the `external-postgres-init.pgsql` script in the previous step.

   To change the passwords, open a terminal to the database using the following command:

   ```
   psql -U blackduck -h <hostname> -p <port> postgres
   ```

   Then run the following three commands in the terminal to change the passwords for the three accounts:

```
ALTER ROLE blackduck WITH PASSWORD 'my_admin_password_here';

ALTER ROLE blackduck_user WITH PASSWORD 'my_user_password_here';

ALTER ROLE blackduck_reporter WITH PASSWORD 'my_reporter_password_here';
```

Hub configuration

1. Ensure that the following Hub environment variables are set properly for your external PostgreSQL instance in the pods.env file:
   - HUB_POSTGRES_ADMIN: "blackduck"
   - HUB_POSTGRES_ENABLE_SSL: "false"

     Because the connection to an external RDS is username/password rather than SSL, make sure to set the HUB_POSTGRES_ENABLE_SSL value to "false".
   - HUB_POSTGRES_HOST: "hostname"
   - HUB_POSTGRES_PORT: "5432"
   - HUB_POSTGRES_USER: "blackduck_user"

   Note: If using a cloud PostgreSQL with a firewall, you may need to allow firewall ingress from 'anywhere' (or at least from a range of IPs that you allocate based on your network egress IP IT information), because you may not know beforehand the IP address of the database that your containers will be connecting to.

2. Like setting proxy passwords, there are two methods to expose the database passwords to your Hub services:
   - In the clear in the config map
   - As Kubernetes secrets in the config map.

     This method is more secure, although more difficult to configure.

Storing passwords in the clear in the config map

Add the passwords for the three database users (**blackduck**, **blackduck_user**, and **blackduck_reporter**) to a config map as follows.

1. Create a /tmp/password file for the **blackduck** (admin) user. The file should have the following content:

```
- apiVersion: v1
   kind: ConfigMap
   metadata:
    name: hpup-admin
   data:
    HUB_POSTGRES_ADMIN_PASSWORD_FILE: |
    <password for admin user here>
```

   Note that this matches the syntax as shown in the kubernetes-external-rds.yml file

2. Run the following command to create the config map:

```
kubectl create -f /tmp/password
```

3. Repeat Steps 1 and 2 for the **blackduck_user**:

   a. Create a /tmp/password file for the **blackduck_user** user. The file should have the following content:

```
- apiVersion: v1

  kind: ConfigMap
  metadata:
   name: hpup-user
  data:
   HUB_POSTGRES_USER_PASSWORD_FILE: |
   <password for user here>
```

   b. Run the following command to create the config map:

```
kubectl create -f /tmp/password
```

4. Repeat Steps 1 and 2 for the **blackduck_reporter**.

Putting DB passwords in Kubernetes secrets and exposing them to Hub services

In the previous section, database passwords were configured in the clear. For added security, you can store the passwords as Kubernetes secrets, and expose those secrets to the appropriate Hub services. The password secrets must be added to the pod specifications for the following Hub services:

- Jobrunner
- Webapp

1. To store a DB password as a secret, place the password in a file, then run the kubectl create secret command:

```
echo "adminpw123" > admin_pwd
echo "userpw123" > user_pwd
echo "reporterpw123" > reporter_pwd

kubectl create secret generic hub_postgres_admin_password --from-
file=./admin_pwd
kubectl create secret generic hub_postgres_user_password --from-
file=./user_pwd
kubectl create secret generic hub_postgres_reporter_password --from-
file=./reporter_pwd
```

   Delete these password files after running these commands.

2. Now the that the secrets have been created in Kubernetes, you must expose the secrets to the Hub services (Webapp and Jobrunner) that require access to it. For example, the following .yml text

exposes the secret to Webapp:

```
image: hub-webapp:4.3.1
env:
- name: HUB_POSTGRES_ADMIN_PASSWORD

 valueFrom:
   secretKeyRef:
     name: hub_postgres_admin_password
     key: admin_pwd

- name: HUB_POSTGRES_USER_PASSWORD
valueFrom:
   secretKeyRef:
     name: hub_postgres_user_password
     key: user_pwd
- name: HUB_POSTGRES_REPORTER_PASSWORD
valueFrom:
   secretKeyRef:
     name: hub_postgres_reporter_password
     key: reporter_pwd
```

3. Add this section of text to the Jobrunner service by replacing `image: hub-webapp:4.3.1` with `image: jobrunner:4.3.1` in the .yml file you used to deploy the Hub (for example, kubernetes-external-rds.yml or kubernes-post-db.yml).

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. For the webserver service, add the secrets by copying their values into the env values for the pod specifications for NGiNX. The sample .yml configuration goes into the hub-nginx image stanza in the .yml file you used to deploy the Hub (for example, kubernetes-external-rds.yml or kubernes-post-db.yml):

   ```
   env:
   - name: WEBSERVER_CUSTOM_CERT_FILE
     valueFrom:
       secretKeyRef:
        name: ws-cust-cert
   ```

3. Add an equivalent stanza for the custom key file:

```
env:

  - name: WEBSERVER_CUSTOM_KEY_FILE
    valueFrom:
      secretKeyRef:
       name: ws-cust-key
```

Scaling job runners

The job runner is the only service that is scalable (it is the only Hub service that can have multiple instances running simultaneously). Job runners can be scaled up or down.

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 Runners. The following example scales back the Job Runner container to a single container:

```
kubectl scale rc jobrunner --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'.

You can define these properties before or after you install the Hub:

- To configure the property before installing the Hub, edit the pod.env file and save your changes.
- To modify the property after installing the Hub, add the environment variables above into the “hub-nginx” image stanza in the .yml file you used to deploy the Hub (for example, kubernetes-external-rds.yml or kubernes-post-db.yml).

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.

These properties are shown in the BLACKDUCK CORS SECTION in the other.env file.

You can define these properties before or after you install the Hub:

- To configure the property before installing the Hub, edit the pod.env file and save your changes.
- To modify the property after installing the Hub, add the environment variables above into the "hub-nginx" image stanza in the .yml file you used to deploy the Hub (for example, kubernetes-external-rds.yml or kubernes-post-db.yml).

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

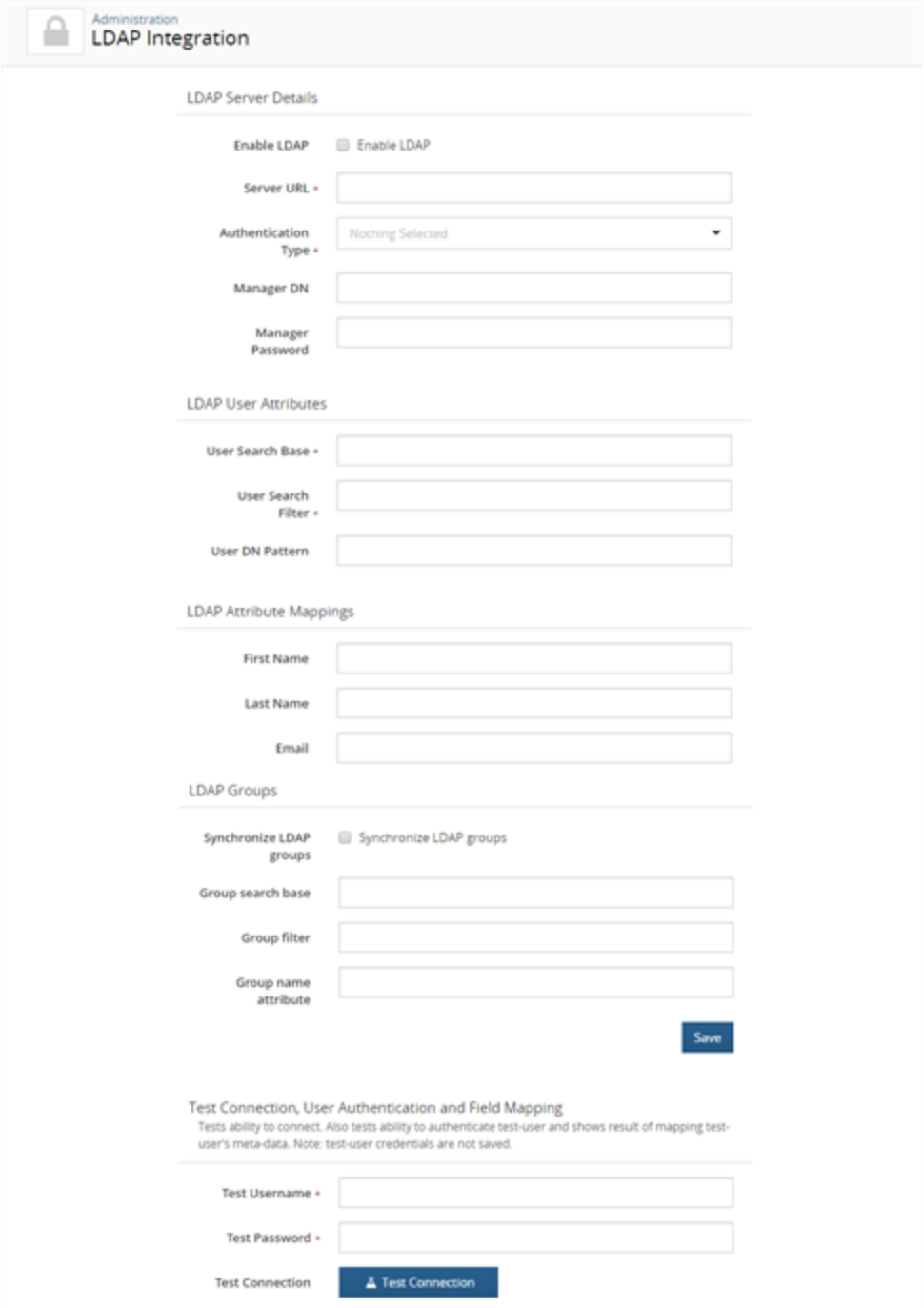
- (required) The user credentials to test the connection to the directory server.

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**.

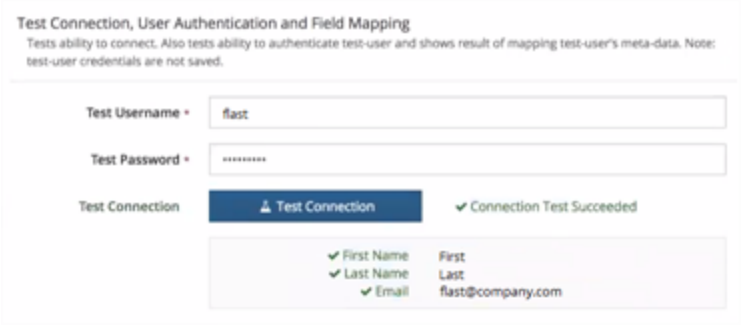
   The Administration page appears.
3. Select **LDAP integration** to display the LDAP Integration page.



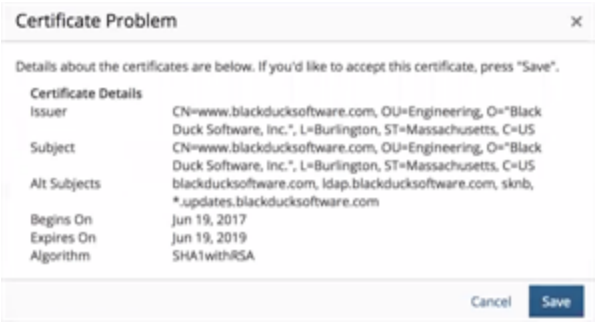
4. Select the **Enable LDAP** option and complete the information in the **LDAP Server Details** and **LDAP User Attributes** sections, as described above. In the **Server URL** field, ensure that you have configured the secure LDAP server: the protocol scheme is ldaps://.

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 Sysadmin user.

For additional SAML information:

- Assertion Consumer Service (ACS): https://host/saml/SSO

Note the following:

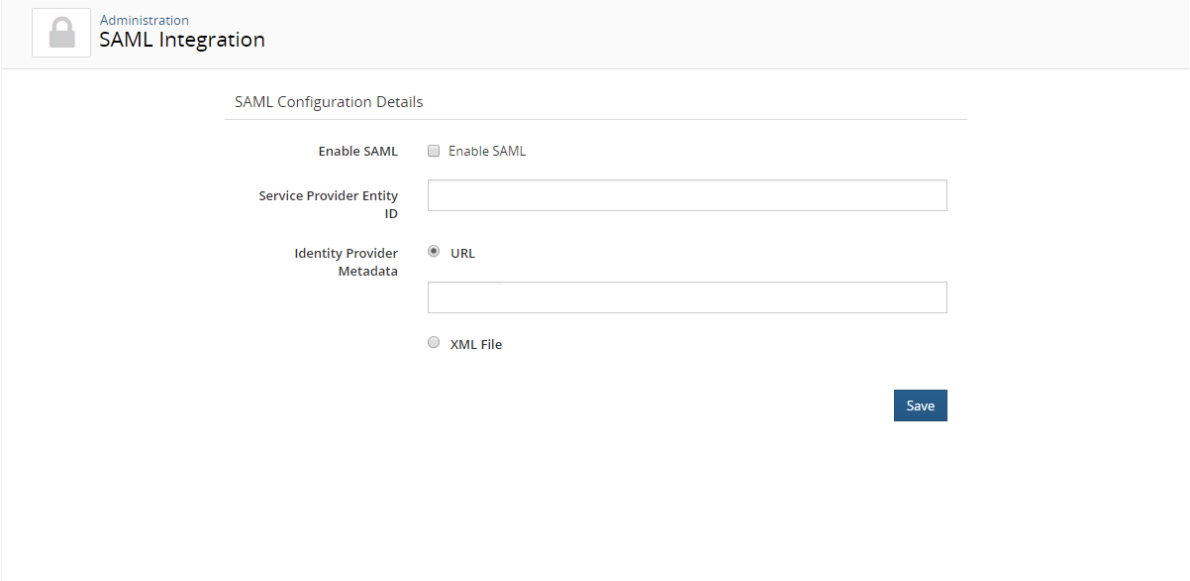
- The Hub is able to sync and obtain an external user's information (Name, FirstName, LastName and Email) if the information is provided in attribute statements. Note that the first and last name values are case-sensitive.
- When logging in with SAML enabled, you are re-directed to your identity provider's login page, not to 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.jsp 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. **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.

c. **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 pods.env (for Kubernetes) or as an environment variable (for 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.

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 an 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 ADD LINK

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-nginx
- hub-logstash
- hub-registration
- hub-solr
- hub-zookeeper

For example, here are the specific commands that must be run to upgrade to the Hub 4.3.1:

```
kubectl set image deployment/cfssl cfssl=cfssl:4.3.1

kubectl set image deployment/documentation
documentation=documentation:4.3.1

kubectl set image deployment/jobrunner jobrunner=jobrunner:4.3.1

kubectl set image deployment/postgres postgres=postgres:4.3.1

kubectl set image deployment/webapp-nginx-logstash webapp=webapp:4.3.1

kubectl set image deployment/webapp-nginx-logstash nginx=nginx:4.3.1

kubectl set image deployment/webapp-nginx-logstash logstash=logstash:4.3.1

kubectl set image deployment/registration registration=registration:4.3.1

kubectl set image deployment/solr solr=solr:4.3.1

kubectl set image deployment/zookeeper zookeeper=zookeeper:4.3.1
```

Appendix A: Debugging a running deployment

This chapter provides information on debugging a Hub on Kubernetes deployment. The procedures can help you determine whether your Kubernetes cluster is working properly.

Viewing running pods

Use the following command to see which pods are running:

```
kubectl get pods
```

You should see output similar to the following:

```
NAME                                    READY     STATUS    RESTARTS   AGE
cfssl-258485687-m3szc                   1/1       Running   0          3h
jobrunner-1397244634-xgcn2              1/1       Running   2          26m
nginx-webapp-logstash-2564656559-6fbq8  3/3       Running   0          26m
postgres-1794201949-tt4gj               1/1       Running   0          3h
registration-2718034894-7brjv           1/1       Running   0          26m
solr-1180309881-sscsl                   1/1       Running   0          26m
zookeeper-3368690434-rnz3m              1/1       Running   0          26m
```

In the above example, there are pods containing a single container each (cfssl, jobrunner, postgres, registration, solr, and zookeeper) and a pod containing three containers (nginx, webapp, logstash).

Executing Docker commands and viewing container log files

You can use the "kubectl exec" command to execute a Docker command inside a Docker container inside a pod. This is especially helpful in viewing log files. The generic syntax is:

```
kubectl exec -t -i <pod_name> -c <container name> <Docker command>
```

For example, to view the log file of the load balancer shown in the previous example, the command is:

```
kubectl exec -t -i nginx-webapp-logstash-2564656559-6fbq8 -c nginx cat
/var/log/nginx/nginx-access.log
```

The command displays the following output:

```
192.168.21.128 - - [12/Jul/2017:18:13:12 +0000] "GET /api/v1/registrations?summary=true&_=1499883191824 HTTP/1.1" 200
192.168.21.128 - - [12/Jul/2017:18:13:12 +0000] "GET /api/internal/logo.png HTTP/1.1" 200 7634 "https://a0145b939671c
10.0.25.32 - - [12/Jul/2017:18:25:42 +0000] "GET / HTTP/1.1" 200 21384 "-" "curl/7.47.0" "-"
```

In another example, we can use the "kubectl logs" command to view the Docker log files (from standard out) for the Hub's Webapp container:

```
kubectl logs nginx-webapp-logstash-2564656559-6fbq8 -c webapp
```

Which displays the following information:

```
2017-07-12 18:13:12,064 [http-nio-8080-exec-4] INFO  com.blackducksoftware.core.regupdate.impl.RegistrationApi - Exec
2017-07-12 18:13:12,071 [http-nio-8080-exec-4] ERROR com.blackducksoftware.core.regupdate.impl.RegistrationApi - Unab
2017-07-12 18:25:42,596 [http-nio-8080-exec-1] INFO  com.blackducksoftware.usermgmt.sso.impl.BdsSAMLEntryPoint - Sing
2017-07-12 18:27:52,670 [scanProcessorTaskScheduler-1] INFO  com.blackducksoftware.scan.bom.scheduler.ScanPurgeJobMor
2017-07-12 18:30:00,059 [job.engine-0] WARN  com.blackducksoftware.job.integration.handler.KbCacheUpdater - KB projec
```

This shows all standard output from Webapp (the Hub's web server). (Although a full description of the content of these log files is beyond the scope of this chapter, a large time period without log message would suggest an issue with the Webapp.)

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 B: Containers

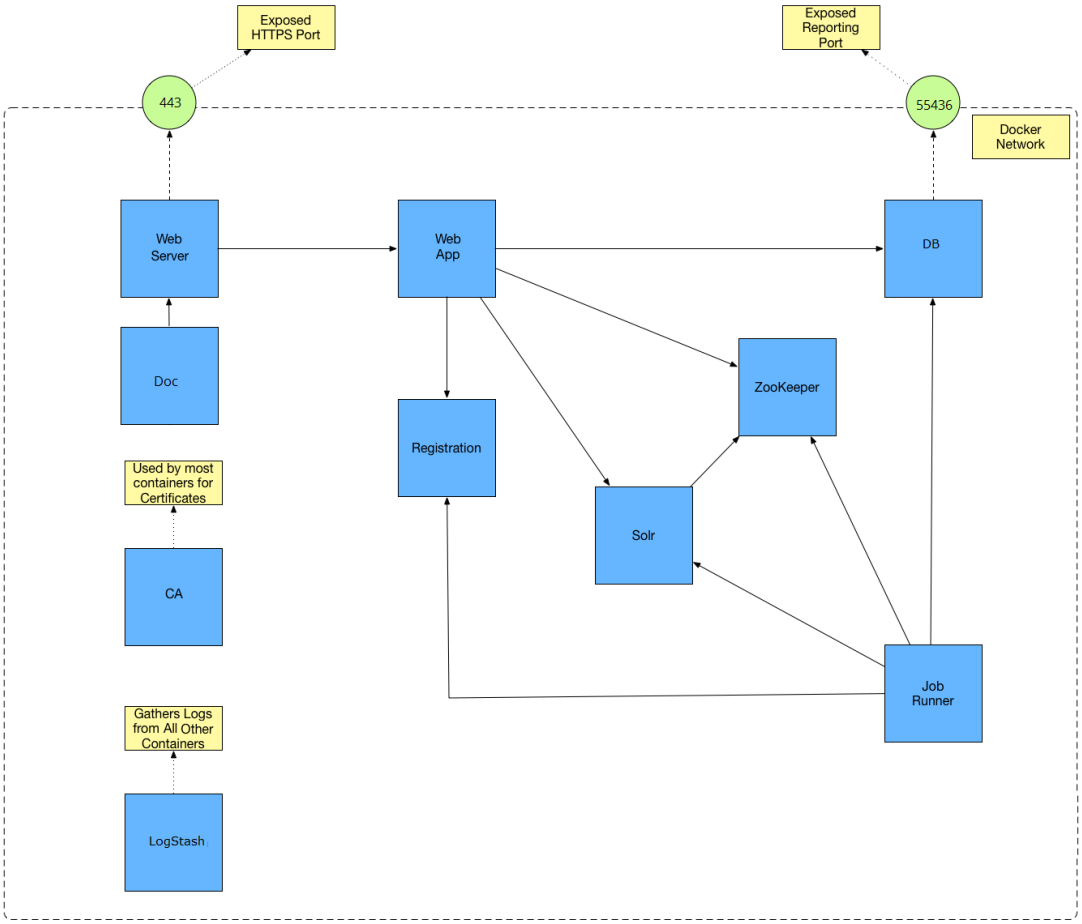
These are the containers within the Docker network that comprise the Hub application:

1. Web App
2. Job Runner App
3. Solr
4. Registration
5. DB

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

6. WebServer
7. Zookeeper
8. LogStash
9. CA
10. Documentation

The following diagram shows the basic relationships among the containers and which ports are exposed outside of the Docker network. It includes a PostgreSQL database container.



This diagram makes no assumptions about which Docker hosts are running which container: it is possible that each container runs on a separate Docker host. All containers are contained within a Docker network. The only two ports exposed outside of the Docker network are the HTTPS port for Hub (via NGiNX) and a read-only database port from Postgres for reporting. All other external communication will go through a proxy or another NGiNX instance. All other communication will be among the containers within the Docker network.

The following tables provide more information on each container.

Web App container

Container Name: Web App	
Image Name	blackducksoftware/hub-webapp:4.3.1
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 8080 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: 4GB • Container memory: 4GB • Container CPU: 1 CPU

Container Name: Web App	
Volumes	log-volume:/opt/blackduck/hub/logs webapp-volume:/opt/blackduck/hub/hub-webapp/security
Environment File	pods.env

Job runner container

Container Name: Job Runner	
Image Name	blackducksoftware/hub-jobrunner:4.3.1
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 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: 4GB • Container memory: 4GB • Container CPU: 1 CPU
Volumes	N/A
Environment File	pods.env

Solr container

Container Name: Solr	
Image Name	blackducksoftware/hub-solr:4.3.1
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 8080 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	N/A
Environment File	N/A

Registration container

Container Name: Registration	
Image Name	blackducksoftware/hub-registration:4.3.1
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 8080 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
Environment File	pods.env

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:4.3.1
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. 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: 2GB • Container CPU: 1 CPU
Volumes	data-volume:/var/lib/postgresql/data
Environment File	N/A

WebServer container

Container Name: WebServer	
Image Name	blackducksoftware/hub-nginx:4.3.1
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 This container exposes port 443 outside of the Docker network.

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

ZooKeeper container

Container Name: Zookeeper	
Image Name	blackducksoftware/hub-zookeeper:4.3.1
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

Container Name: Zookeeper	
Volumes	N/A
Environment File	N/A

LogStash container

Container Name: LogStash	
Image Name	blackducksoftware/hub-logstash:4.3.1
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.
Constraints	• Default max Java heap size: 1GB • Container memory: 1GB • Container CPU: Unspecified
Volumes	log-volume:/var/lib/logstash/data
Environment File	N/A

CA container

Container Name: CA	
Image Name	blackducksoftware/hub-cfssl:4.3.1
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
Environment File	N/A

Documentation container

Container Name: CA	
Image Name	blackducksoftware/hub-cfssl:4.3.1
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 ■ webserver The documentation container must expose port 8080 to other containers that link to it.
Constraints	■ Default Max Java Heap Size: 512MB ■ Container Memory: 512MB ■ Container CPU: unspecified
Volumes	N/A
Environment File	N/A