BLACKDUCK | Hub

Installation Guide

Version 4.2.1

This edition of the *Installation Guide* refers to version 4.2.1 of the Black Duck Hub.

This document created or updated on Tuesday, October 3, 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.
Installation Guide	hub_install.pdf	Contains information about installing and upgrading the Hub.
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

This document provides instructions for installing Black Duck Hub in a Docker environment.

Hub Architecture

The Black Duck Hub is deployed as a set of Docker containers. "Dockerizing" the Hub so that different components are containerized allows third-party orchestration tools such as Compose or Swarm to manage all individual containers.

The Docker architecture brings these significant improvements to the Hub:

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

See Docker containers, for more information on the Docker containers that comprise the Hub application.

Visit the Docker website: https://www.docker.com/ for more information on Docker. To obtain installation information, go to https://docs.docker.com/engine/installation/.

Components hosted on Black Duck servers

The components hosted on Black Duck servers are:

- **Registration server**: Used to validate the Hub license.
- **Black Duck KnowledgeBase server**: Cloud-based version of the Black Duck KnowledgeBase (KB) that contains many of the most popular open source projects in the KB. Hosting the Black Duck KB in the cloud means that the Hub can display the most up-to-date information about open source software (OSS) projects without requiring regular updates on your host machine.

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 (either based on the AppMgr architecture or based on the Docker architecture).

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 CPUs
- 16GB RAM (20 GB RAM for Docker Swarm)
- 250 GB of free disk space for the database and other Hub containers
- Commensurate space for database backups

The descriptions of each container document the individual requirements for each container if it will be 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.

Black Duck recommends monitoring disk utilization on the Hub servers to prevent disks from reaching capacity which could cause issues with the Hub.

Docker requirements

Docker Version

The Hub installation supports Docker Version 17.03.x or 17.06.x (CE or EE).

Supported orchestration tools

- Docker Swarm (Preferred Method) - a clustering and scheduling tool for Docker containers. With Docker Swarm, you can manage a cluster of Docker nodes as a single virtual system.

  There are two restrictions when using the Hub in Docker Swarm:

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

    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 host.

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

  Note: For scalability, Black Duck recommends running the Hub on a single node Swarm deployment.

- Docker Compose - a tool for running multi-container Docker applications.

  The minimum supported version of docker-compose must be able to read Docker Compose 2.1 files.

The distribution content for each orchestration tool is described here.

Operating systems

The preferred operating systems for installing the Hub in a Docker environment are:

- 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

In addition, Black Duck will support other Linux operating systems that support Docker version 17.03.x (CE or EE).

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 60.0.3112.113 (Official Build) (64-bit)
- Firefox 55.0.3 (64-bit), 46.0.1
- Internet Explorer 11.0.43 (KB4025252)
- Microsoft Edge 40.15063.0.0
- Microsoft EdgeHTML 14.14393
- Safari 10.1.2 (11603.3.8)

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.

PostgreSQL versions

For the Hub version 4.2.1, 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 4.2.1.

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

Screen resolutions

The Hub UI requires a minimum horizontal screen resolution of 1280. The following screen resolutions have been tested using the minimum horizontal requirement:

- 1280 x 720
- 1280 x 768
- 1280 x 800
- 1280 x 960
- 1280 x 1024

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

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.

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 PostgreSQL instance: the Hub supports using Amazon Relational Database Service (RDS) for the external 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.

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.

Use the hub-proxy.env file to configure your proxy settings.

Configuring your NGiNX server to work with the Hub

If you have an NGINX server acting as an HTTPS server/proxy in front of the Hub, you must modify the

NGINX configuration file so that the NGINX server passes the correct headers to the Hub. The Hub then generates the URLs that use HTTPS.

Note: Only one service on the NGINX server can use https port 443.

To pass the correct headers to the Hub, edit the `location` block in the nginx.config configuration file to:

```
location / {
   client_max_body_size 1024m;
   proxy_pass http://127.0.0.1:8080;
   proxy_pass_header X-Host;
   proxy_set_header Host $host:$server_port;
   proxy_set_header X-Real-IP $remote_addr;
   proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
   proxy_set_header X-Forwarded-Proto $scheme;
}
```

If the X-Forwarded-Prefix header is being specified in a proxy server/load balancer configuration, edit the `location` block in the nginx.config configuration file:

```
location/prefixPath {
   proxy_set_header X-Forwarded-Prefix "/prefixPath";
}
```

To scan files successfully, you must use the **context** parameter in the Hub Scanner or include it in the **Hub Server URL** field in the Hub Scanner 2.0 - Beta.

Note: Although these instructions apply to an NGINX server, similar configuration changes would need to be made for any type of proxy server.

If the proxy server will rewrite requests to the Hub, let the proxy server administrator know that the following HTTP headers can be used to preserve the original requesting host details.

HTTP Header	Description
X-Forwarded-Host	Tracks the list of hosts that were re-written or routed to make the request. The original host is the first host in the comma-separated list. **Example:** `X-Forwarded-Host: "10.20.30.40,my.example, 10.1.20.20"`
X-Forwarded-Port	Contains a single value representing the port used for the original request.

	Example: `X-Forwarded-Port: "9876"`
X-Forwarded-Proto	Contains a single value representing the protocol scheme used for the original request. **Example:** `X-Forwarded-Proto: "https"`
X-Forwarded-Prefix	Contains a prefix path used for the original request. **Example:** `X-Forwarded-Prefix: "prefixPath"` To successfully scan files, you must use the **context** parameter

Amazon services

You can:

- Install the Hub on Amazon Web Services (AWS)

  Refer to your AWS documentation and your AMI 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.
Docker requirements	
☐	You have ensured that your system meets the docker 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.

Installation media

The installation media is available on Github (https://github.com/blackducksoftware/hub).

Downloaded the orchestration files:

```
wget https://github.com/blackducksoftware/hub/raw/master/archives/hub-docker-4.2.1.tar
```

Unpack the Hub `.tar` file:

```
tar xvf hub-docker-4.2.1.tar
```

Distributions

Each orchestration tool has a directory with the files you need to install or upgrade the Hub.

- `docker-compose.dbmigrate.yml`: Docker Compose file used to migrate the PostgreSQL database when using the database container provided by the Hub.
- `docker-compose.externaldb.yml`: Docker Compose file used with an external PostgreSQL database.
- `docker-compose.yml`: Docker Compose file when using the database container provided by the Hub.
- `external-postgres-init.pgsql`: PostgresSQL.sql file used to configure an external PostgreSQL database.
- `hub-postgres.env`: Environment file to configure an external PostgreSQL database.
- `hub-proxy.env`: Environment file to configure proxy settings.
- `hub-webserver.env`: Environment file to configure web server settings.

In the `bin` directory:

- `hub_create_data_dump.sh`: Script used to back up the PostgreSQL database when using the database container provided by the Hub.
- `hub_db_migrate.sh`: Script used to migrate 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.
- `system_check.sh`: Script used to gather your Hub system information to send to Customer Support.

Installing the Hub

This section provides instructions for new installations of the Hub using:

- Docker Swarm
- Docker Compose

Installing the Hub using Docker Swarm

These instructions only apply to installing the Hub using Docker Swarm.

Prior to installing the Hub, determine if there are any web server, proxy, or database settings that need to be configured.

To install the Hub, you may need to be a user in the docker group, a root user, or have `sudo` access.

Note: These instructions are for new installations of the Hub. Refer to Chapter 6 for more information about upgrading the Hub.

- To install the Hub with the PostgreSQL database container:

```
docker swarm init
docker stack deploy -c docker-compose.yml hub
```

Note: Use the version of the `docker-compose.yml` file located in the docker-swarm directory.

The `docker swarm init` command creates a single-node swarm.

- To install the Hub with an external PostgreSQL instance:

```
docker swarm init
docker stack deploy -c docker-compose.externaldb.yml hub
```

Note: Use the version of the `docker-compose.externaldb.yml` file located in the docker-swarm directory.

The `docker swarm init` command creates a single-node swarm.

Note: There are some versions of Docker where if the images live in a private repository, docker stack will not pull them unless the following flag is added to the above commands: `--with-registry-auth`.

You can confirm that the installation was successful by running the `docker ps` command to view the status of each container. A "healthy" status indicates that the installation was successful. Note that the containers may be in a "starting" state for a few minutes post-installation.

Once all of the containers for 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 web server settings and then you can access the Hub by entering the following:

https://hub.example.com

A dialog box appears requesting a registration key. Enter the key provided to you to access the Hub.

Installing the Hub using Docker Compose

These instructions only apply to installing the Hub using Docker Compose.

Prior to installing the Hub, determine if there are any web server, proxy, or database settings that need to be configured.

To install the Hub, you may need to be a user in the docker group, a root user, or have `sudo` access.

Note: These instructions are for new installations of the Hub. Refer to Chapter 6 for more information about upgrading the Hub.

- To install the Hub with the PostgreSQL database container:

```
docker-compose -f docker-compose.yml -p hub up -d
```

Note: Use the version of the `docker-compose.yml` file located in the docker-compose directory.

- To install the Hub with an external PostgreSQL instance:

```
docker-compose -f docker-compose.externaldb.yml -p hub up -d
```

Note: Use the version of the `docker-compose.yml` file located in the docker-compose directory.

The Black Duck Hub is installed.

You can confirm that the installation was successful by running the `docker ps` command to view the status of each container. A "healthy" status indicates that the installation was successful. Note that the containers may be in a "starting" state for a few minutes post-installation.

Once all of the containers for 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 web server settings and then you can access the Hub by entering the following:

https://hub.example.com

A dialog box appears requesting a registration key. Enter the key provided to you to access the Hub.

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.

Providing your Hub system information to Customer Support

Customer Support may ask you to provide them with information regarding your Hub installation, such as system statistics and environmental or network information. To make it easier for you to quickly obtain this information, Black Duck provides a script, `system_check.sh`, which you can use to collect this information. The script outputs this information to a file, system_check.txt, located in your working directory, which you can then send to Customer Support.

Providing your Hub system information when using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

The `system_check.sh` script is located in the `docker-swarm/bin` directory:

```
./bin/system_check.sh
```

Note that to run this script, you may need to be a user in the docker group, a root user, or have `sudo` access.

Providing your Hub system information when using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

The `system_check.sh` script is located in the `docker-compose/bin` directory:

```
./bin/system_check.sh
```

Note that to run this script, you may need to be a user in the docker group, a root user, or have `sudo` access.

Chapter 4: Post-installation tasks

Optionally after installing the Hub, you can:

- Replace the existing self-signed certificate for the Web Server with a custom certificate.
- Access log files.
- Scale job runners.
- Configure secure LDAP.
- Configure Single Sign-On (SSO).
- Configure the report database password.
- Provide access to the REST APIs from a non-Hub server.

You can also modify the configuration settings after installing the Hub.

Using custom certificates

The Web Server container has a self-signed certificate obtained from Docker. You may want to replace this certificate with a custom certificate-key pair.

Using custom certificates with Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

1. Use the docker secret command to tell Docker Swarm the certificate and key by using WEBSERVER_CUSTOM_CERT_FILE and WEBSERVER_CUSTOM_KEY_FILE. The name of the secret must include the stack name. In the following example, the stack name is 'hub':

   ```
   docker secret create hub_WEBSERVER_CUSTOM_CERT_FILE <certificate file>

   docker secret create hub_WEBSERVER_CUSTOM_KEY_FILE <key file>
   ```

   1. Add the secret to the services section of the Webserver service:

      ```
      secrets: [WEBSERVER_CUSTOM_CERT_FILE, WEBSERVER_CUSTOM_KEY_
      FILE]
      ```

      Note: If you are using Docker 17.06, you must add text such as the following to the end of the compose file:

```
secrets:
 WEBSERVER_CUSTOM_CERT_FILE:
    external:
      name: "hub_WEBSERVER_CUSTOM_CERT_FILE"
 WEBSERVER_CUSTOM_KEY_FILE:
    external:
      name: "hub_WEBSERVER_CUSTOM_KEY_FILE"
```

2. The healthcheck property in the webserver service must point to the new certificate from the secret:

```
healthcheck:
test: [CMD, /usr/local/bin/docker-healthcheck.sh,
'https://localhost:8443/health-checks/liveness',
/run/secrets/WEBSERVER_CUSTOM_CERT_FILE]
```

Using custom certificates with Docker Compose

These instructions only apply when running the Hub using Docker Compose.

1. Create a file named WEBSERVER_CUSTOM_CERT_FILE with the certificate file.
2. Create a file named WEBSERVER_CUSTOM_KEY_FILE with the key file.
3. Mount a directory that contains both files to `/run/secrets` in the Webserver container by editing the volumes section of the `docker-compose.yml` or `docker-compose.externaldb.yml` file located in the docker-compose directory.

```
webserver
 image: blackducksoftware/hub-nginx:4.2.1
 ports: ['443:8443']
 env_file:hub-webserver.env
 links: [webapp, cfssl]
 volumes: ['webserver-
volume:/opt/blackduck/hub/webserver/security',
'/directory/where/the/cert-key-files/are:/run/secrets']
```

4. Start the Webserver container.

Accessing log files

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

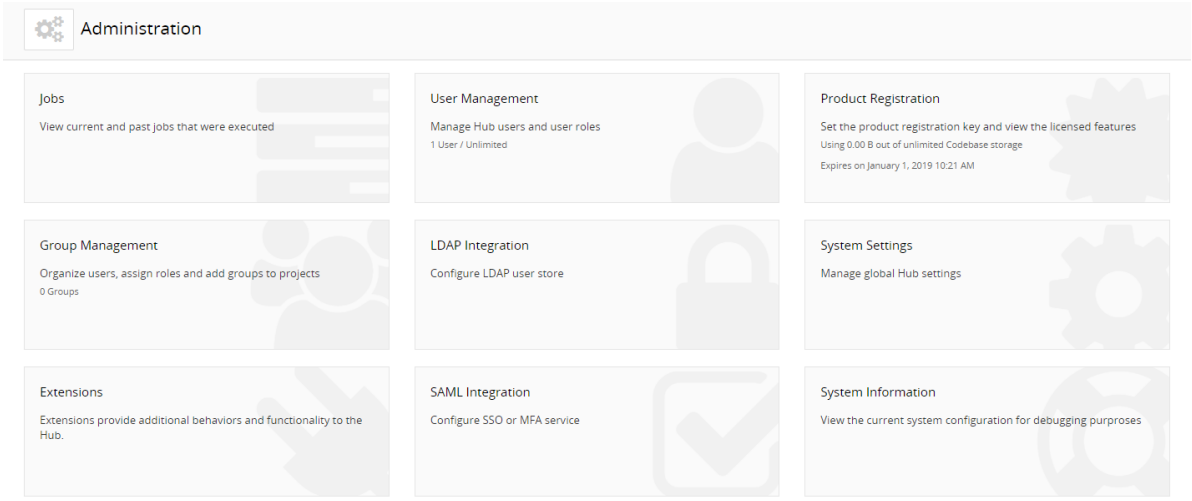
Users with the System Administrator role can download a zipped file that contains the current log files.

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 (

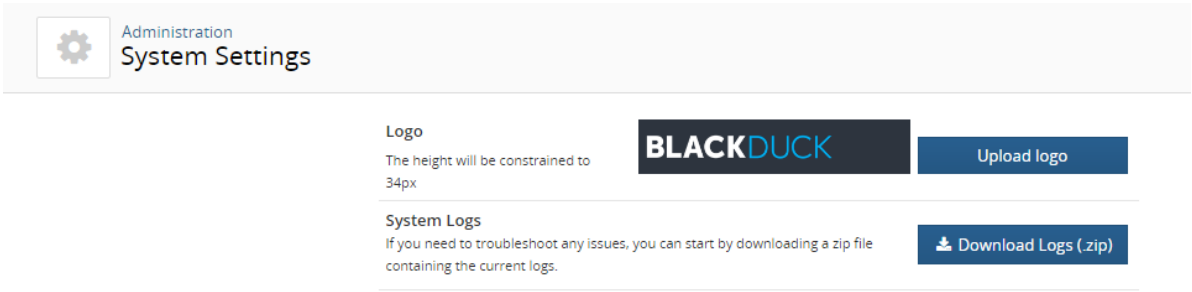
) and select **Administration**.

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.

Obtaining logs

To obtain logs from the containers:

```
docker cp <logstash container ID>:/var/lib/logstash/data logs/
```

where '*logs/*' is a local directory where the logs will be copied into.

Viewing log files for a container

Use the docker-compose `logs` command to view all logs:

```
docker-compose logs
```

Scaling job runners

The Job Runner container can be scaled.

Scaling job runners when using Docker Swarm

These instructions only apply to scaling the Job Runner container when running the Hub using Docker Swarm.

You may need to be a user in the docker group, a root user, or have `sudo` access to run the following command.

This example adds a second Job Runner container:

```
docker service scale hub_jobrunner=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:

```
docker service scale hub_jobrunner=1
```

Scaling job runners when using Docker Compose

These instructions only apply to scaling the Job Runner container when running the Hub using Docker Compose.

You may need to be a user in the docker group, a root user, or have `sudo` access to run the following command.

This example adds a second Job Runner container:

```
docker-compose -f docker-compose.yml -p hub up --scale jobrunner=2 -d
```

Note: Use the version of the `docker-compose.yml` file located in the docker-compose directory.

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:

```
docker-compose -f docker-compose.yml -p hub up --scale jobrunner=1 -d
```

Note: Use the version of the `docker-compose.yml` file located in the docker-compose directory.

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

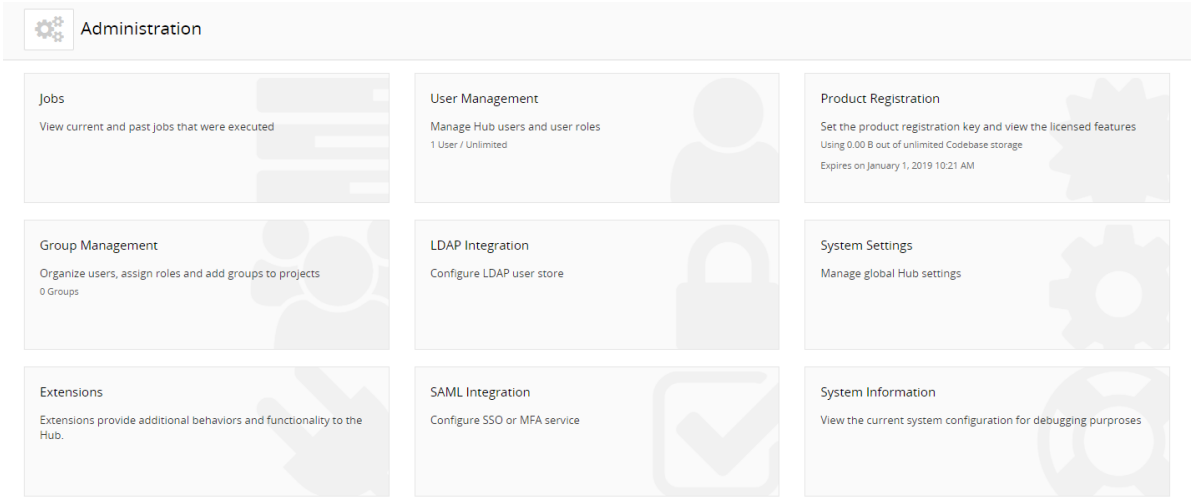
- (required) The user credentials to test the connection to the directory server.

Importing the server certificate

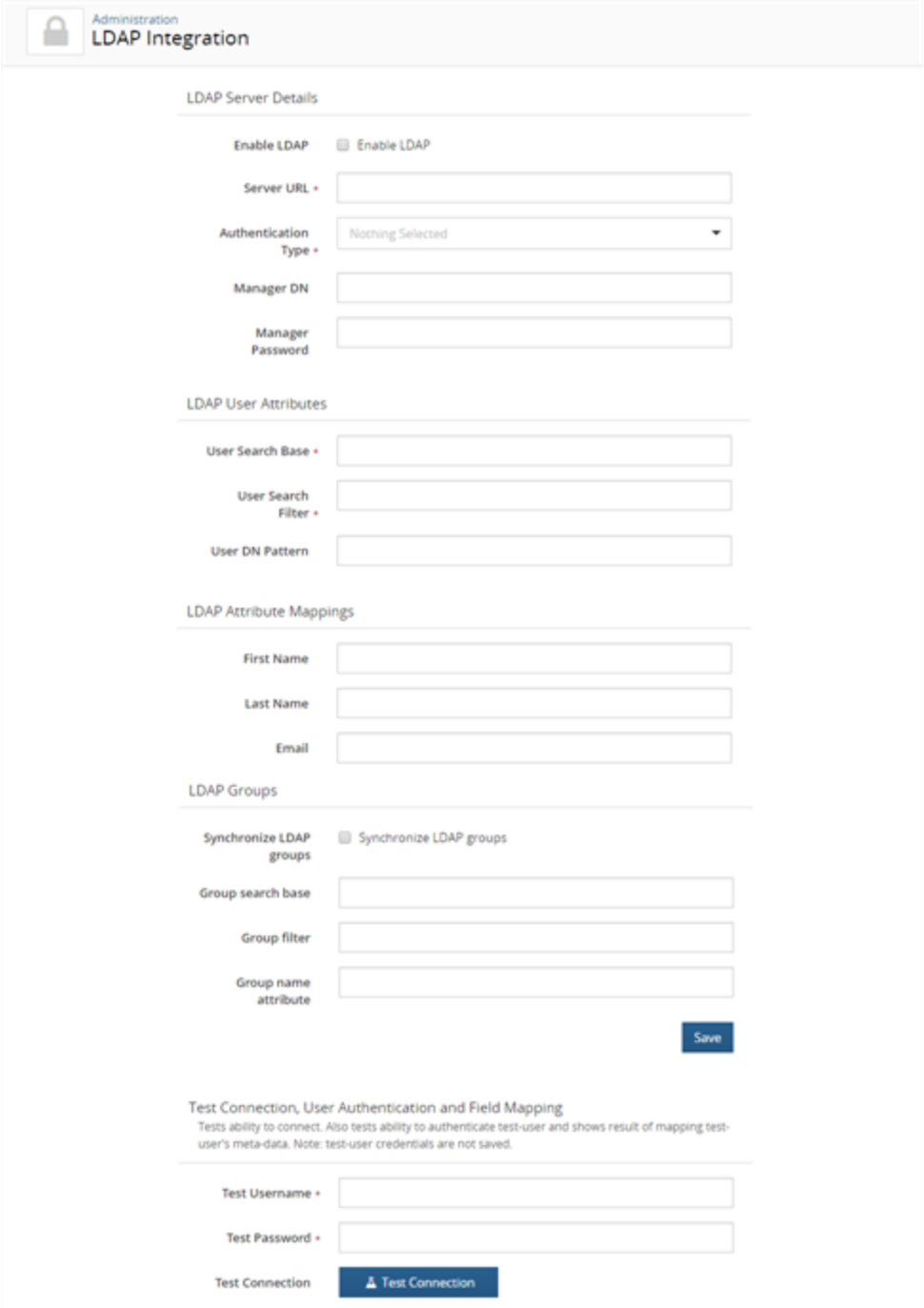
To import the server certificate

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

   The Administration page appears.



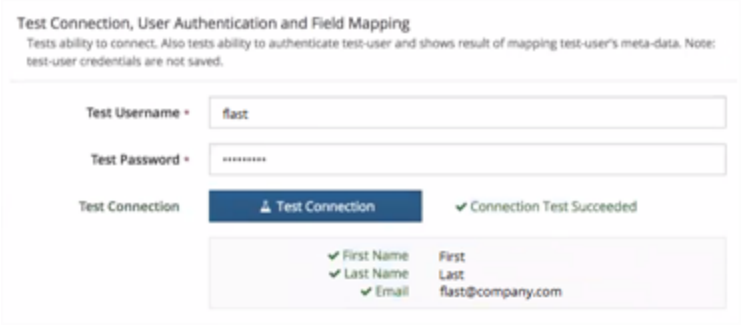
3. Select **LDAP integration** to display the LDAP Integration page.



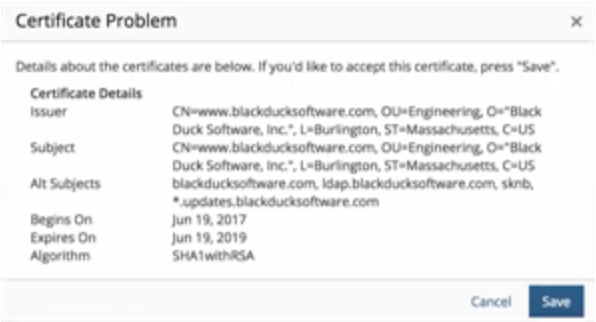
4. Select the **Enable LDAP** option and complete the information in the **LDAP Server Details** and **LDAP User Attributes** sections, as described above. In the **Server URL** field, ensure that you have configured the secure LDAP server: the protocol scheme is ldaps://.

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

If you add a custom Hub web application trust store, use these methods for specifying an LDAP trust store password.

The methods you can use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

There are three methods for specifying an LDAP trust store password when using Docker Swarm.

- Use the docker secret command to tell Docker Swarm the password by using LDAP_TRUST_STORE_PASSWORD_FILE. The name of the secret must include the stack name. 'HUB' is the stack name in this example:

  ```
  docker secret create HUB_LDAP_TRUST_STORE_PASSWORD_FILE <file containing password>
  ```

  Add the password secret to the services section of the webapp service:

  ```
  secrets:
   - LDAP_TRUST_STORE_PASSWORD_FILE
  ```

  Note that if you are using Docker 17.06, you must add text such as the following to the end of the compose file:

  ```
  secrets:
    LDAP_TRUST_STORE_PASSWORD_FILE:
        external:
          name: "HUB_LDAP_TRUST_STORE_PASSWORD_FILE"
  ```

- Mount a directory that contains a file called LDAP_TRUST_STORE_PASSWORD_FILE to `/run/secrets` by editing the volumes section for webapp services in the `docker-compose.yml` or `docker-compose.externaldb.yml` file located in the docker-swarm directory.

  ```
  volumes: ['log-volume:/opt/blackduck/hub/logs', 'webapp-volume:/opt/blackduck/hub/hub-webapp/security', '/directory/where/files/are:/run/secrets']
  ```

- Specify an environment variable called LDAP_TRUST_STORE_PASSWORD that contains the password.

Docker Compose

These instructions only apply when running the Hub using Docker Compose.

There are two methods for specifying an LDAP trust store password when using Docker Compose.

- Mount a directory that contains a text file called LDAP_TRUST_STORE_PASSWORD_FILE to `/run/secrets` by editing the volumes section for webapp services in the `docker-compose.yml` or `docker-compose.externaldb.yml` file located in the docker-compose directory.

  ```
  volumes: ['log-volume:/opt/blackduck/hub/logs', 'webapp-volume:/opt/blackduck/hub/hub-webapp/security', '/directory/where/files/are:/run/secrets']
  ```

- Specify an environment variable called LDAP_TRUST_STORE_PASSWORD that contains the LDAP trust store password.

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

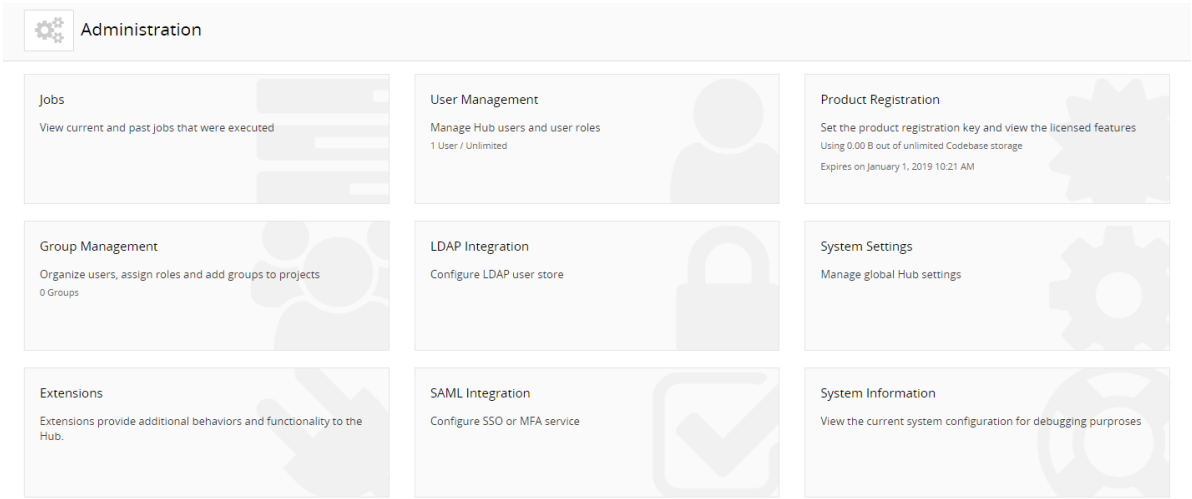
Note the following:

- When logging in with SAML enabled, you are re-directed to your identity provider's login page, not to The Hub's login page.
- Single Logout is not supported and therefore the **Log Out** menu option from the Hub is disabled: you must logout from the Identity Provider.
- 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.

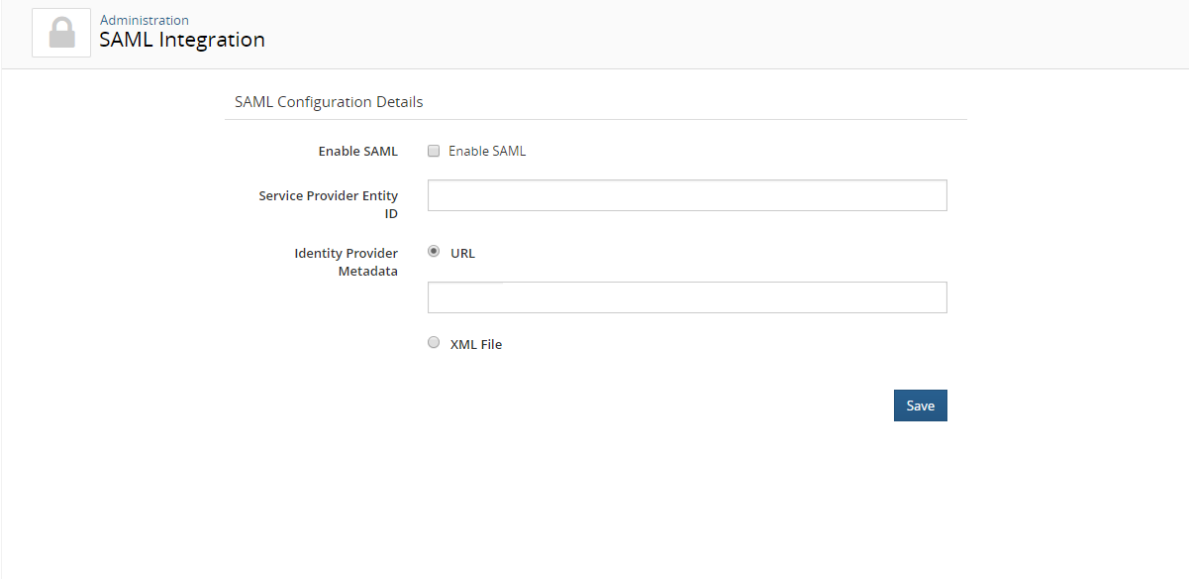
To enable single sign-on using SAML

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**.

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

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.

Configuring the report database password

This section provides instructions on configuring the report database password. The method you use

depends on whether you are running the Hub using Docker Swarm or Docker Compose.

Configuring the report database password when using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

Use the `hub_reportdb_changepassword.sh` script, located in the `docker-swarm/bin` directory 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 Docker host 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
```

Configuring the report database password when using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

Use the `hub_reportdb_changepassword.sh` script, located in the `docker-compose/bin` directory 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 Docker host 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
```

Providing access to 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. The method you can use depends on whether you are running the Hub using Docker Swarm or Docker Compose.

Accessing the REST APIs from a non-Hub server when using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

To enable access to the REST APIs from a non-Hub server, 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 configure these properties, edit the hub-proxy.env file, located in the docker-swarm directory.

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

- To set these properties *before* installing the Hub, edit the file and save your changes.
- To modify these properties *after* installing the Hub, modify the properties and then redeploy the services in the stack by entering:

```
docker stack deploy -c docker-compose.yml hub
```

Note: Use the version of the `docker-compose.yml` file located in the docker-swarm directory.

Accessing the REST APIs from a non-Hub server when using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

To enable access to the REST APIs from a non-Hub server, 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 configure these properties, edit the hub-proxy.env file, located in the docker-compose directory.

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

- To set these properties *before* installing the Hub, edit the file and save your changes.
- To modify these properties *after* installing the Hub, do the following:

  1. Stop the containers.

     ```
     docker-compose -p hub stop
     ```

  2. Open the file and edit the properties.
  3. Reconnect the containers.

     ```
     docker-compose -p hub up
     ```

Chapter 5: Uninstalling the Hub

Follow these instructions to uninstall the Hub. The methods you use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Uninstalling the Hub using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

Use either of these methods to uninstall the Hub:

- Stop and remove the containers and remove the volumes.

  ```
  docker stack rm hub
  ```

- Stop and remove the containers but keep the volumes. For example:

  ```
  docker volume prune
  ```

  Caution: This command removes *all* unused volumes: volumes not referenced by *any* container are removed. This includes unused volumes not used by other applications.

Note that the PostgreSQL database is not backed up. Use these instructions to back up the database.

Uninstalling the Hub using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

To uninstall the Hub, do one of the following:

- Stop and remove the containers and remove the volumes.

  ```
  docker-compose -p hub down -v
  ```

- Stop and remove the containers but keep the volumes. For example:

  ```
  docker-compose -p hub down
  ```

Note that the PostgreSQL database is not backed up. Use these instructions to back up the database.

Chapter 6: Upgrading the Hub

The Hub supports upgrading to any available version, giving you the ability to jump multiple versions in a single upgrade.

The upgrade instructions depend on your previous version of the Hub:

- AppMgr architecture
- Single-container AppMgr architecture
- Multi-container Docker

Installation media

The installation media is available on Github (https://github.com/blackducksoftware/hub).

Downloaded the orchestration files:

```
wget https://github.com/blackducksoftware/hub/raw/master/archives/hub-docker-4.2.1.tar
```

Unpack the Hub `.tar` file:

```
tar xvf hub-docker-4.2.1.tar
```

Upgrading from the AppMgr architecture

This section describes how to upgrade from a previous version of the Hub based on the AppMgr architecture to the multi-container Docker architecture.

The instructions you use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Using Docker Swarm

These instructions describe how to upgrade from a previous version of the Hub based on the AppMgr architecture when running the Hub using Docker Swarm.

Upgrading to the multi-container Docker architecture consists of:

1. Migrating your PostgreSQL database.

   This is an optional step if you want to retain your existing database data.

2. Upgrading the Hub.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Restoring 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.

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-swarm directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker stack deploy -c docker-compose.dbmigrate.yml hub
   ```

   Note that there are some versions of Docker where if the images live in a private repository, docker stack will not pull them unless the following flag is added to the above command:

   ```
   --with-registry-auth
   ```

2. After the DB container has started, run the migration script located in the docker-swarm directory. This script restores the data from the existing database dump file.

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

You can now upgrade to the multi-image Docker version of the Hub.

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

1. If you ran the `setup-autostart.sh` script in your previous AppMgr version of the Hub, you will need to remove the 'iptable' entries that were created by that script. As a root user, cd to the directory where you installed the Hub, for example, `/opt/blackduck/hub/appmgr/bin` and run the `iptables-redirect.sh` script with the `delete` parameter:

   ```
   ./iptables-redirect.sh delete
   ```

   Note that you can safely run this script If you are unsure if autostart was configured as this script makes no changes if the previous AppMgr version of the Hub was not configured for autostart.

2. If you are installing the Hub on the same server that had the AppMgr version of the Hub installed on it:

   a. Run the `uninstall.sh` script to remove old files:

   ```
   /opt/blackduck/hub/appmgr/bin/uninstall.sh
   ```

   b. As a root user or with sudo access, remove the autostart file. The `uninstall.sh` script states the location of the file at the end of the script run. For example:

   ```
   rm -rf /etc/init.d/bds-hub-controller
   ```

3. Run one of the following commands, located in the docker-swarm directory, using the files in the newer version of the Hub. The command depends on whether you are using the DB container or an external PostgreSQL instance:

   - Using the DB container: `docker stack deploy -c docker-compose.yml hub`
   - Using an external PostgreSQL database: `docker stack deploy -c docker-compose.externaldb.yml hub`

Using Docker Compose

These instructions describe how to upgrade from a previous version of the Hub based on the AppMgr architecture when running the Hub using Docker Compose.

Upgrading to the multi-container Docker architecture consists of:

1. Migrating your PostgreSQL database.

   This is an optional step if you want to retain your existing database data.

2. Upgrading the Hub.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Restoring 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.

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-compose directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker-compose -f docker-compose.dbmigrate.yml -p hub up -d
   ```

2. After the DB container has started, run the migration script located in the docker-compose directory. This script restores the data from the existing database dump file.

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

After stopping the containers, you can upgrade to the multi-image Docker version of the Hub.

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

1. If you ran the `setup-autostart.sh` script in your previous AppMgr version of the Hub, you will need to remove the 'iptable' entries that were created by that script. As a root user, cd to the directory where you installed the Hub, for example, `/opt/blackduck/hub/appmgr/bin` and run the `iptables-redirect.sh` script with the `delete` parameter:

   ```
   ./iptables-redirect.sh delete
   ```

   Note that you can safely run this script If you are unsure if autostart was configured as this script makes no changes if the previous AppMgr version of the Hub was not configured for autostart.

2. If you are installing the Hub on the same server that had the AppMgr version of the Hub installed on it:

   a. Run the `uninstall.sh` script to remove old files:

   ```
   /opt/blackduck/hub/appmgr/bin/uninstall.sh
   ```

   b. As a root user or with sudo access, remove the autostart file. The `uninstall.sh` script states the location of the file at the end of the script run. For example:

   ```
   rm -rf /etc/init.d/bds-hub-controller
   ```

3. Run one of the following commands using the files in the newer version of the Hub located in the docker-compose directory. The command depends on whether you are using the DB container or an external PostgreSQL instance:

   - Using the DB container: `docker-compose -f docker-compose.yml -p hub up -d`
   - Using an external PostgreSQL instance: `docker-compose -f docker-compose.externaldb.yml -p hub up -d`

Upgrading from a single-container AppMgr Hub

This section describes how to upgrade from a previous version of the Hub based on the single-container AppMgr architecture to the multi-container Docker architecture. The instructions you use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Using Docker Swarm

These instructions describe how to upgrade from a previous version of the Hub based on the single-

container AppMgr architecture when running the Hub using Docker Swarm.

Upgrading to the multi-container Docker architecture consists of:

1. Migrating your PostgreSQL database.

   This is an optional step if you want to retain your existing database data.

2. Upgrading the Hub.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Restoring the data.

To back up the PostgreSQL database

1. Run the following command to create a PostgreSQL dump file:

   ```
   docker exec -it <containerid or name> pg_dump -U blackduck -Fc -f
   /tmp/bds_hub.dump bds_hub
   ```

2. Copy the dump file out of the container by running the following command:

   ```
   docker cp <containerid>:<path to dump file in container> .
   ```

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-swarm directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker stack deploy -c docker-compose.dbmigrate.yml hub
   ```

   Note that there are some versions of Docker where if the images live in a private repository, docker stack will not pull them unless the following flag is added to the above command:

   ```
   --with-registry-auth
   ```

2. After the DB container has started, run the migration script located in the docker-swarm directory. This script restores the data from the existing database dump file.

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

You can now upgrade to the multi-image Docker version of the Hub.

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

1. Run one of the following commands, located in the docker-swarm directory, using the files in the newer version of the Hub. The command depends on whether you are using the DB container or an external PostgreSQL instance:

   - Using the DB container: `docker stack deploy -c docker-compose.yml hub`
   - Using an external PostgreSQL database: `docker stack deploy -c docker-compose.externaldb.yml hub`

Using Docker Compose

These instructions describe how to upgrade from a previous version of the Hub based on the single-container AppMgr architecture when running the Hub using Docker Compose.

Upgrading to the multi-container Docker architecture consists of:

1. Migrating your PostgreSQL database.

   This is an optional step if you want to retain your existing database data.

2. Upgrading the Hub.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Restoring the data.

To back up the PostgreSQL database

1. Run the following command to create a PostgreSQL dump file:

   ```
   docker exec -it <containerid or name> pg_dump -U blackduck -Fc -f
   /tmp/bds_hub.dump bds_hub
   ```

2. Copy the dump file out of the container by running the following command:

   ```
   docker cp <containerid>:<path to dump file in container> .
   ```

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-compose directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker-compose -f docker-compose.dbmigrate.yml -p hub up -d
   ```

2. After the DB container has started, run the migration script located in the docker-compose

directory. This script restores the data from the existing database dump file.

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

After stopping the containers, you can upgrade to the multi-image Docker version of the Hub.

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

1. Run one of the following commands using the files in the newer version of the Hub located in the docker-compose directory. The command depends on whether you are using the DB container or an external PostgreSQL instance:
   - Using the DB container: `docker-compose -f docker-compose.yml -p hub up -d`
   - Using an external PostgreSQL instance: `docker-compose -f docker-compose.externaldb.yml -p hub up -d`

Upgrading from an existing Docker architecture

This section describes how to upgrade the Hub.

The instructions you use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

To upgrade from a previous version of the Hub:

1. Migrate your PostgreSQL database.

   As the PostgreSQL database version has been upgraded to version 9.6.x in 4.2.1, you must migrate your database prior to upgrading.

   The data migration will temporarily require additional free disk space at approximately 2.5 times your original database volume size to hold the database dump and the new 4.2 database volume. As a rule-of-thumb, if the volume upon which your database resides is at least 60% free, there should be enough disk space.

2. Upgrade the Hub.

Note: The method to configure custom SSL certificates for NGiNX changed in 4.1.0. If you are upgrading from version 4.0.0 or 4.0.1 and you had configured custom SSL certificates for NGiNX, you will need to reconfigure them.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Bringing down the Hub containers.
3. Restoring the data.

Note: If your Hub instance was configured to use an external database (like Amazon RDS), the recommended approach is to migrate your data to a 9.6 instance of PostgreSQL and configure your system to point to that instance. If an administrator attempts to perform an upgrade to Hub 4.2 on a system that is connected to a non-9.6 PostgreSQL database, the application will fail to start, however the data remains safe.

To back up the PostgreSQL database that is automatically installed with the Hub

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.

To bring down the Hub containers

1. Run the following command to bring down the Hub containers which removes the current stack that has the previous version of the Hub running:

   ```
   docker stack rm hub
   ```

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-swarm directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker stack deploy -c docker-compose.dbmigrate.yml hub
   ```

   Note that there are some versions of Docker where if the images live in a private repository, docker stack will not pull them unless the following flag is added to the above command:

   ```
   --with-registry-auth
   ```

2. After the DB container has started, run the migration script located in the docker-swarm directory.

This script restores the data from the existing database dump file.

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

You can now upgrade the Hub.

Upgrading the Hub

To upgrade the Hub:

1. Run the following command using the files included in the newer version of the Hub located in the docker-swarm directory. The command depends on whether you are using the DB container or an external PostgreSQL instance:
   - Using the DB container: `docker stack deploy -c docker-compose.yml hub`
   - Using an external PostgreSQL instance: `docker stack deploy -c docker-compose.externaldb.yml hub`

Using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

To upgrade from a previous version of the Hub:

1. Migrate your PostgreSQL database.

   As the PostgreSQL database version has been upgraded to version 9.6.x in 4.2.1, you must migrate your database prior to upgrading.

   The data migration will temporarily require additional free disk space at approximately 2.5 times your original database volume size to hold the database dump and the new 4.2 database volume. As a rule-of-thumb, if the volume upon which your database resides is at least 60% free, there should be enough disk space.

2. Upgrade the Hub.

Note: The method to configure custom SSL certificates for NGiNX changed in 4.1.0. If you are upgrading from version 4.0.0 or 4.0.1 and you had configured custom SSL certificates for NGiNX, you will need to reconfigure them.

Migrating your PostgreSQL database

To use your existing PostgreSQL data, you must migrate the database data which consists of:

1. Backing up the original PostgreSQL database.
2. Bringing down the Hub containers.
3. Restoring the data.

Note: If your Hub instance was configured to use an external database (like Amazon RDS), the recommended approach is to migrate your data to a 9.6 instance of PostgreSQL and configure your system to point to that instance. If an administrator attempts to perform an upgrade to Hub 4.2 on a system that is connected to a non-9.6 PostgreSQL database, the application will fail to start, however the data remains safe.

To back up the PostgreSQL database that is automatically installed with the Hub

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.

To bring down the Hub containers

1. Run the following command to bring down the Hub containers which removes the current stack that has the previous version of the Hub running:
   - Using the DB container: `docker-compose -f docker-compose.yml -p hub down`
   - Using an external PostgreSQL instance: `docker-compose -f docker-compose.externaldb.yml -p hub down`

To restore the PostgreSQL data

1. Use the `docker-compose.dbmigrate.yml` file located in the docker-compose directory. It starts the containers and volumes needed to migrate the database.

   ```
   docker-compose -f docker-compose.dbmigrate.yml -p hub up -d
   ```

2. After the DB container has started, run the migration script located in the docker-compose directory. This script restores the data from the existing database dump file.

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

You can now upgrade the Hub.

Upgrading the Hub

To upgrade the Hub:

1. Stop the existing containers:

   ```
   docker-compose -p hub down
   ```

2. Run the following command using the files included in the newer version of the Hub located in the docker-compose directory. The command depends on whether you are using the DB container or an external PostgreSQL instance:

- Using the DB container: `docker-compose -f docker-compose.yml -p hub up -d`
- Using an external PostgreSQL instance: `docker-compose -f docker-compose.externaldb.yml -p hub up -d`

Appendix A: Configuration settings

Use this appendix to manage your configuration settings and configure an external PostgreSQL instance.

Managing configuration settings using Docker Swarm

These instructions only apply to managing your configuration settings when using Docker Swarm.

There are environment files located in the docker-swarm directory which you can use to configure web server, proxy, and external PostgreSQL settings:

- hub-webserver.env
- hub-proxy.env
- hub-postgres.env

To configure the settings:

- To set configuration settings *before* installing the Hub, edit the file as described below and save your changes.
- To modify existing settings *after* installing the Hub, modify the settings and then redeploy the services in the stack by entering:
  - `docker stack deploy -c docker-compose.yml hub` if using the DB container
  - `docker stack deploy -c docker-compose.externaldb.yml hub` if using an external PostgreSQL instance

Note: Use the version of the `docker-compose.yml` or `docker-compose.externaldb.yml` file located in the docker-swarm directory.

Configuring Web server settings

Edit the hub-webserver.env file to:

- Configure the hostname.
- Configure the host port.

- Disable IPv6.

Configuring the hostname

Edit the hub-webserver.env file to configure the hostname so the certificate host name matches. The environment variable has the service name as the default value.

When the web server starts up, it generates an HTTPS certificate if certificates are not configured. You must specify a value for the PUBLIC_HUB_WEBSERVER_HOST environment variable to tell the web server the hostname it will listen on so that the hostnames can match. Otherwise, the certificate will only have the service name to use as the hostname. This value should be changed to the publicly-facing hostname that users will enter in their browser in order to access Hub. For example:

```
PUBLIC_HUB_WEBSERVER_HOST=blackduck-docker01.dc1.lan
```

Configuring the host port

You can configure a different value for the host port which, by default, is 443.

To configure the host port

1. Modify the host port value defined in the following files.

   In the docker-compose.yml or docker-compose.externaldb.yml file, edit the first value shown in `ports: ['443:8443']` to the new port value.

   ```
   webserver:
    image: blackducksoftware/hub-nginx:4.2.1
    ports: ['443:8443']
   ```

   For example, to change the port to 8443:

   ```
   webserver:
    image: blackducksoftware/hub-nginx:4.2.1
    ports: ['8443:8443']
   ```

2. Edit the PUBLIC_HUB_WEBSERVER_PORT value in the hub-webserver.env file to the new port value. For example:

   ```
   PUBLIC_HUB_WEBSERVER_PORT=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 environment variable to 1.

Configuring Proxy settings

Edit the hub-proxy.env file to configure proxy settings. You will need to configure these settings if a proxy is required for external internet access.

There are three containers that need access to services hosted by Black Duck:

- Registration
- Job runner
- Web App

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

The following services require the proxy password:

- Web App
- Registration
- Job Runner

There are three methods for specifying a proxy password:

- Mount a directory that contains a text file called HUB_PROXY_PASSWORD_FILE to `/run/secrets`. This is the most secure option.
- Specify an environment variable called HUB_PROXY_PASSWORD that contains the proxy password.
- Use the docker secret command to create a secret called HUB_PROXY_PASSWORD_FILE as described below:

    1. Use the docker secret command to tell Docker Swarm the secret. The name of the secret must include in the stack name. In the following example, the stack name is 'hub':

        ```
        docker secret create hub_HUB_PROXY_PASSWORD_FILE <file containing password>
        ```

    2. Add to the services section of the Web App, Registration, and Job Runner services:

        ```
        secrets:
         - HUB_PROXY_PASSWORD_FILE
        ```

        Note that if you are using Docker 17.06, you must add text such as the following to the end of the compose file:

```
secrets:
  HUB_PROXY_PASSWORD_FILE:
     external:
       name: "hub_HUB_PROXY_PASSWORD_FILE"
```

You can use the hub-proxy.env file to specify an environment variable if it is not specified in a separate mounted file or secret:

1. Remove the pound sign (#) located in front of HUB_PROXY_PASSWORD so that it is no longer commented out.
2. Enter the proxy password.
3. Save the file.

Configuring the Hub session timeout

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

To specify a different value, edit the hub-proxy.env file by adding the HUB_WEBAPP_SESSION_TIMEOUT property and specifying the new timeout value in number of seconds.

For example, to specify a timeout value of one hour (3600 seconds), enter:

HUB_WEBAPP_SESSION_TIMEOUT=3600

Managing configuration settings using Docker Compose

These instructions only apply to managing your configuration settings when using Docker Compose.

There are environment files located in the docker-compose directory. Use these files to configure web server, proxy, and external PostgreSQL settings:

- hub-webserver.env
- hub-proxy.env
- hub-postgres.env

To configure the settings:

- To set configuration settings *before* installing the Hub, edit the file as described below and save your changes.
- To modify existing settings *after* installing the Hub, do the following:
  1. Stop the containers:

     ```
     docker-compose -p hub stop
     ```

  2. Open the file and edit the settings.
  3. Reconnect the containers:

     ```
     docker-compose -p hub up
     ```

Configuring Web server settings

Edit the hub-webserver.env file to:

- Configure the hostname.
- Configure the host port.
- Disable IPv6.

Configuring the hostname

Edit the hub-webserver.env file to configure the hostname so the certificate host name matches. The environment variable has the service name as the default value.

When the web server starts up, it generates an HTTPS certificate if certificates are not configured. You must specify a value for the PUBLIC_HUB_WEBSERVER_HOST environment variable to tell the web server the hostname it will listen on so that the hostnames can match. Otherwise, the certificate will only have the service name to use as the hostname. This value should be changed to the publicly-facing hostname that users will enter in their browser in order to access Hub. For example:

```
PUBLIC_HUB_WEBSERVER_HOST=blackduck-docker01.dc1.lan
```

Configuring the host port

You can configure a different value for the host port which, by default, is 443.

To configure the host port

1. Modify the host port value defined in the following files.

   In the docker-compose.yml or docker-compose.externaldb.yml file, edit the first value shown in `ports: ['443:8443']` to the new port value.

   ```
   webserver:
    image: blackducksoftware/hub-nginx:4.2.1
    ports: ['443:8443']
   ```

   For example, to change the port to 8443:

   ```
   webserver:
    image: blackducksoftware/hub-nginx:4.2.1
    ports: ['8443:8443']
   ```

2. Edit the PUBLIC_HUB_WEBSERVER_PORT value in the hub-webserver.env file to the new port value. For example:

   ```
   PUBLIC_HUB_WEBSERVER_PORT=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 environment variable to 1.

Configuring Proxy settings

Edit the hub-proxy.env file to configure proxy settings. You will need to configure these settings if a proxy is required for external internet access.

There are three containers that need access to services hosted by Black Duck:

- Registration
- Job runner
- Web App

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

The following services require the proxy password:

- Web App
- Registration
- Job Runner

There are two methods for specifying a proxy password:

- Mount a directory that contains a text file called HUB_PROXY_PASSWORD_FILE to `/run/secrets`. This is the most secure option.
- Specify an environment variable called HUB_PROXY_PASSWORD that contains the proxy password.

You can use the hub-proxy.env file to specify an environment variable if it is not specified in a separate mounted file or secret:

1. Remove the pound sign (#) located in front of HUB_PROXY_PASSWORD so that it is no longer commented out.
2. Enter the proxy password.
3. Save the file.

Configuring the Hub session timeout

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

To specify a different value, edit the hub-proxy.env file by adding the HUB_WEBAPP_SESSION_TIMEOUT

property and specifying the new timeout value in number of seconds.

For example, to specify a timeout value of one hour (3600 seconds), enter:

HUB_WEBAPP_SESSION_TIMEOUT=3600

Configuring an external PostgreSQL instance

This section describes how to configure and external PostgreSQL instance.

The instructions you use depend on whether you are running the Hub using Docker Swarm or Docker Compose.

Configuring an external PostgreSQL instance using Docker Swarm

These instructions only apply when running the Hub using Docker Swarm.

The Hub supports using an external PostgreSQL instance managed by Amazon Relational Database Service (RDS). Be sure that you have configured the instance as described below prior to installing or upgrading the Hub.

To configure an external PostgreSQL instance

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, located in the docker-swarm directory, to install the Hub, to create users, databases, and other necessary items. For example:

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

3. Using your preferred PostgreSQL administration tool, 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.

4. Edit the hub-postgres.env environment file, as described here, to specify the database connection parameters:

Parameter	Description
HUB_POSTGRES_ENABLE_SSL	Forces the use of SSL in database connections. As Amazon RDS automatically uses SSL, this must be set to "false".
HUB_POSTGRES_HOST	Hostname of the server with the PostgreSQL instance.
HUB_POSTGRES_PORT	Database port to connect to for the PostgreSQL instance.

Parameter	Description
HUB_POSTGRES_USER	Database username. By default, this is set to **blackduck_user**.
HUB_POSTGRES_ADMIN	Database administrator. By default, this is set to **blackduck**.

5. Provide the **blackduck** and **blackduck_user** passwords to the Hub:

   a. Create a file named HUB_POSTGRES_USER_PASSWORD_FILE with the password for the **blackduck_user** user.

   b. Create a file named HUB_POSTGRES_ADMIN_PASSWORD_FILE with the password for the **blackduck** user.

   c. Mount a directory that contains both files to `/run/secrets` in both the Web App and Job runner containers by editing the `docker-compose.externaldb.yml` file.

   Instead of Steps 5a-c, you can use the docker secret command to create a secret called HUB_POSTGRES_USER_PASSWORD_FILE and a secret called HUB_POSTGRES_ADMIN_PASSWORD_FILE.

   a. Use the docker secret command to tell Docker Swarm the secret. The name of the secret must include the stack name. In the following example, the stack name is 'hub':

   ```
   docker secret create hub_HUB_POSTGRES_USER_PASSWORD_FILE <file
   containing password>

   docker secret create hub_HUB_POSTGRES_ADMIN_PASSWORD_FILE <file
   containing password>
   ```

   b. Add the password secret to the services section of the Web App and Job Runner services:

   ```
   secrets:
    - HUB_POSTGRES_USER_PASSWORD_FILE
    - HUB_POSTGRES_ADMIN_PASSWORD_FILE
   ```

   Note that if you are using Docker 17.06, you must add text such as the following to the end of the compose file:

   ```
   secrets:
     HUB_POSTGRES_USER_PASSWORD_FILE:
        external:
          name: "hub_HUB_POSTGRES_USER_PASSWORD_FILE"
     HUB_POSTGRES_ADMIN_PASSWORD_FILE:
        external:
          name: "hub_HUB_POSTGRES_ADMIN_PASSWORD_FILE"
   ```

6. Install or upgrade the Hub.

Configuring an external PostgreSQL instance using Docker Compose

These instructions only apply when running the Hub using Docker Compose.

The Hub supports using an external PostgreSQL instance managed by Amazon Relational Database

Service (RDS). Be sure that you have configured the instance as described below prior to installing or upgrading the Hub.

To configure an external PostgreSQL instance

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`, located in the docker-compose directory, 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. Using your preferred PostgreSQL administration tool, 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.

4. Edit the hub-postgres.env environment file, as described here, to specify the database connection parameters:

Parameter	Description
HUB_POSTGRES_ENABLE_SSL	Forces the use of SSL in database connections. As Amazon RDS automatically uses SSL, this must be set to "false".
HUB_POSTGRES_HOST	Hostname of the server with the PostgreSQL instance.
HUB_POSTGRES_PORT	Database port to connect to for the PostgreSQL instance.
HUB_POSTGRES_USER	Database username. By default, this is set to **blackduck_user**.
HUB_POSTGRES_ADMIN	Database administrator. By default, this is set to **blackduck**.

5. Provide the **blackduck** and **blackduck_user** passwords to the Hub:

   a. Create a file named HUB_POSTGRES_USER_PASSWORD_FILE with the password for the **blackduck_user** user.

   b. Create a file named HUB_POSTGRES_ADMIN_PASSWORD_FILE with the password for the **blackduck** user.

   c. Mount a directory that contains both files to `/run/secrets` in both the Web App and Job runner containers by editing the `docker-compose.externaldb.yml` file.

6. Install or upgrade the Hub.

Appendix B: Docker containers

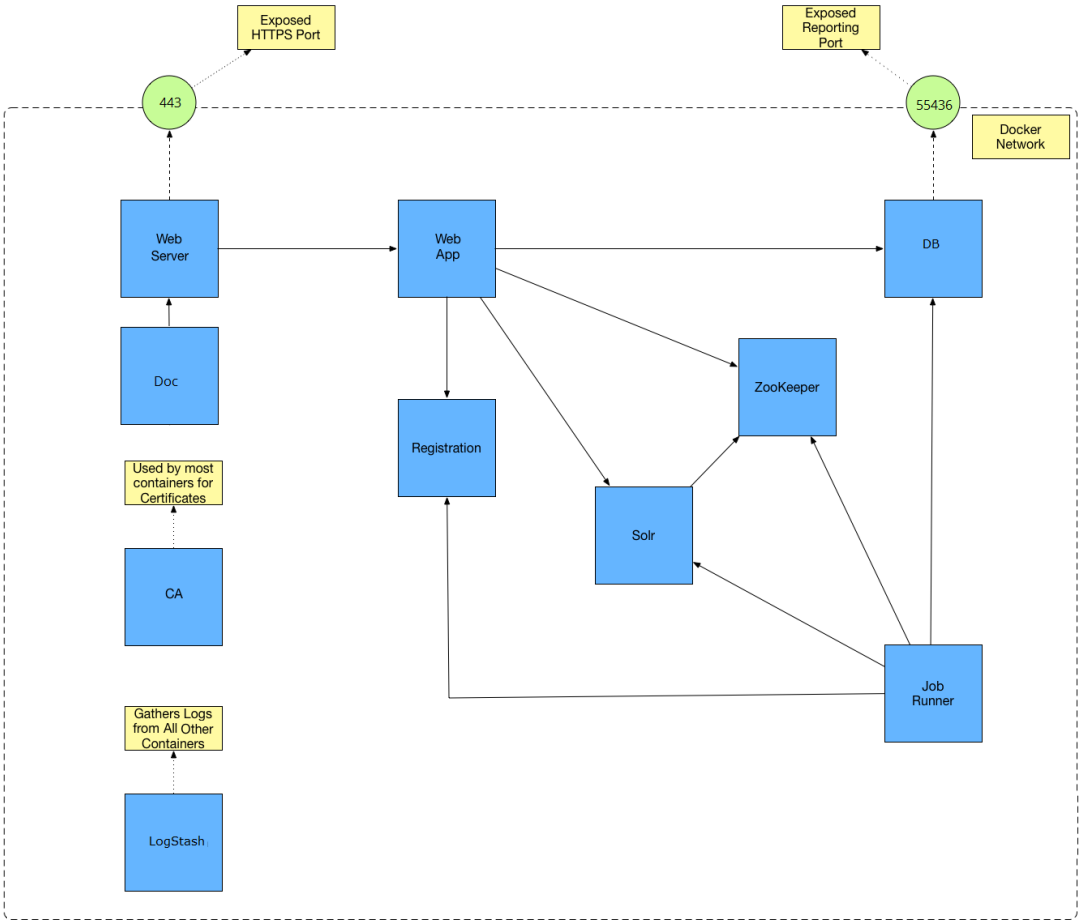
These are the containers within the Docker network that comprise the Hub application:

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:4.2.1
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	hub-proxy.env

Job runner container

Container Name: Job Runner	
Image Name	blackducksoftware/hub-jobrunner:4.2.1
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	hub-proxy.env

Solr container

Container Name: Solr	
Image Name	blackducksoftware/hub-solr:4.2.1
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:4.2.1
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	hub-proxy.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:4.2.1
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:4.2.1
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	hub-webserver.env

ZooKeeper container

Container Name: Zookeeper	
Image Name	blackducksoftware/hub-zookeeper:4.2.1
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:4.2.1
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:4.2.1
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:4.2.1
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