



Docker Swarmを使用したBlack Duck SCAのインストール

Black Duck SCA 2024.10.0

Copyright ©2024 by Black Duck.

All rights reserved.本マニュアルの使用はすべて、Black Duck Software, Inc. とライセンス所有者間の使用許諾契約に準拠します。本ドキュメントのいかなる部分も、Black Duck Software, Inc.の書面による許諾を受けることなく、どのような形態または手段によっても、複製・譲渡することが禁じられています。

Black Duck、Know Your Code、およびBlack Duckロゴは、米国およびその他の国におけるBlack Duck Software, Inc.の登録商標です。Black Duck Code Center、Black Duck Code Sight、Black Duck Hub、Black Duck Protex、Black Duck Suiteは、Black Duck Software, Inc.の商標です。他の商標および登録商標はすべてそれぞれの所有者が保有しています。

09-12-2024

目次

まえがき.....	7
Black Duck documentation.....	7
カスタマサポート.....	8
Black Duck コミュニティ.....	8
トレーニング.....	8
Black Duck 包括性と多様性に関する声明.....	9
Black Duck セキュリティへの取り組み.....	9
 1. 概要.....	10
Black Duckサーバーでホストされるコンポーネント.....	10
 2. インストールの計画.....	11
使用する前に.....	11
新規インストール.....	11
以前のバージョンのからのアップグレード: Black Duck.....	11
ハードウェア要件.....	11
Dockerの要件.....	12
Dockerバージョン.....	13
RHELへのDockerエンジンのインストール.....	13
オペレーティングシステム.....	13
ソフトウェア要件.....	13
ネットワーク要件.....	14
データベース要件.....	15
PostgreSQLのバージョン.....	15
一般的な移行プロセス.....	16
Swarmでの移行の概要.....	16
プロキシサーバーの要件.....	17
連携するNGiNXサーバーの設定: Black Duck.....	17
Amazonサービス.....	18
ポートに関するその他の情報.....	18
キープアライブ設定の構成.....	19
ナレッジベースフィードバックサービス.....	19
ユーザーエージェント解析.....	19
フィードバックサービスの無効化.....	20
 3. インストール: Black Duck.....	21
インストールファイル.....	21
GitHubページからダウンロードする場合.....	21
wgetコマンドを使用してダウンロードする場合.....	22
配布.....	22
インストール: Black Duck.....	23
高速スキャン: Black Duck.....	25
 4. 管理タスク.....	27

環境ファイルと変数.....	27
重複している構成表の検出.....	27
Bearerトークンの有効期限の変更.....	27
KBMATCH_SENDFPATHパラメータについて.....	28
プロキシサーバー経由でのAPIドキュメントへのアクセス.....	28
Black Duckサーバー以外からのREST APIへのアクセス.....	28
ダッシュボードの更新間隔の構成.....	29
証明書の管理.....	30
カスタム証明書の使用.....	30
ナレッジベースからのコンポーネント移行データの取得.....	32
移行の記録の有効化.....	32
移行データの保持.....	32
APIエンドポイント.....	32
無視されたコンポーネントをレポートに含める.....	33
セキュリティ保護されたLDAPの設定.....	33
LDAP情報の取得.....	33
サーバー証明書のインポート.....	34
LDAP信頼ストアのパスワード.....	35
ログファイルとヒートマップデータのダウンロード.....	36
ログファイルの表示.....	36
デフォルトのメモリ制限の変更.....	37
webappコンテナのデフォルトのメモリ制限の変更.....	37
jobrunnerコンテナのデフォルトのメモリ制限の変更.....	38
scanコンテナのデフォルトのメモリ制限の変更.....	39
binaryscannerコンテナのデフォルトのメモリ制限の変更.....	40
bomengineコンテナのデフォルトのメモリ制限の変更.....	40
logstashのホスト名の変更.....	41
マップされていないコードの場所のクリーンアップ.....	41
cron文字列を使用したスキャンページジョブのスケジュール.....	42
スタックしている構成表イベントのクリア.....	42
上書きファイルの使用.....	42
分析の構成: Black Duck.....	42
外部PostgreSQLインスタンスの構成.....	43
既存の外部データベースのPostgreSQLユーザー名の変更.....	46
プロキシ設定の構成.....	47
レポートデータベースのパスワードの設定.....	49
job runner、scan、bomengine、およびbinaryscannerコンテナのスケーリング.....	49
bomengineコンテナのスケーリング.....	49
job runnerコンテナのスケーリング.....	49
scanコンテナのスケーリング.....	50
binaryscannerコンテナのスケーリング.....	50
シングルサインオンのSAMLの設定.....	50
ソースファイルのアップロード.....	52
起動または停止 Black Duck.....	53
起動 Black Duck.....	53
上書きファイル使用時のBlack Duckの起動.....	53
シャットダウン Black Duck.....	54
ユーザーセッションタイムアウトの構成.....	54
カスタマサポートへのお客様のBlack Duckシステム情報の提供.....	55
デフォルトのsysadminユーザーについて.....	56
Black Duckレポート遅延の構成.....	56
コンテナのタイムゾーンの構成.....	56
デフォルトの使用法の変更.....	56
アップロードされたjsonld/bdioファイルbdio2のマッチタイプ.....	58

Black DuckコンテナのユーザーIDのカスタマイズ.....	58
Webサーバー設定の構成.....	59
ホスト名の構成.....	59
ホストポートの構成.....	60
IPv6の無効化.....	60
UTF8文字エンコードを使用した構成表レポートの作成.....	60
スキャン監視.....	60
HTMLレポートのダウンロードサイズの設定.....	61
KBライセンス更新ジョブおよびセキュリティ更新ジョブの構成.....	61
シークレット暗号化の構成: Black Duck.....	61
シードの生成.....	62
バックアップシードの構成.....	63
シークレットローテーションの管理.....	64
Blackduck Storageのカスタムボリュームの構成.....	64
複数ボリュームの設定.....	64
ボリューム間の移行.....	67
レポートのストレージボリュームの設定.....	68
jobrunnerスレッドプールの設定.....	68
構成表サポートのための高速スキャンの設定.....	69
長時間実行しているジョブのしきい値を変更.....	69
SCM統合の有効化.....	69
自動スキャンRetryヘッダーの設定.....	69
階層的サブプロジェクトのライセンス競合設定.....	70
JWT 公開/秘密鍵ペアのプロビジョニング.....	70
セッション トークンの無効化を設定.....	71
 5. アンインストール: Black Duck.....	 72
 6. アップグレード Black Duck.....	 73
インストールファイル.....	73
GitHubページからダウンロードする場合.....	73
wgetコマンドを使用してダウンロードする場合.....	73
監査イベントテーブルの未使用の行をパージする移行スクリプト.....	74
既存のDockerアーキテクチャからのアップグレード.....	75
データのバックアップと復元.....	77
 7. Dockerコンテナ.....	 78
Authenticationコンテナ.....	79
Binaryscannerコンテナ.....	80
Bomengineコンテナ.....	81
CAコンテナ.....	81
DBコンテナ.....	82
Documentationコンテナ.....	83
統合コンテナ.....	84
Jobrunnerコンテナ.....	84
Logstashコンテナ.....	85
Matchengineコンテナ.....	86
Rabbitmqコンテナ.....	86
Redisコンテナ.....	87
Registrationコンテナ.....	88

目次

ReversingLabs コンテナ	89
Scanコンテナ	89
ストレージコンテナ	90
Webappコンテナ	91
Webserverコンテナ	92

まえがき

Black Duck documentation

Black Duckのドキュメントは、オンラインヘルプと次のドキュメントで構成されています：

タイトル	ファイル	説明
リリースノート	release_notes.pdf	新機能と改善された機能、解決された問題、現在のリリースおよび以前のリリースの既知の問題に関する情報が記載されています。
Docker Swarmを使用したBlack Duckのインストール	install_swarm.pdf	Docker Swarmを使用したBlack Duckのインストールとアップグレードに関する情報が記載されています。
Kubernetesを使用したBlack Duckのインストール	install_kubernetes.pdf	Kubernetesを使用したBlack Duckのインストールとアップグレードに関する情報が記載されています。
OpenShiftを使用したBlack Duckのインストール	install_openshift.pdf	OpenShiftを使用したBlack Duckのインストールとアップグレードに関する情報が記載されています。
使用する前に	getting_started.pdf	初めて使用するユーザーにBlack Duckの使用法に関する情報を提供します。
スキャンベストプラクティス	scanning_best_practices.pdf	スキャンのベストプラクティスについて説明します。
SDKを使用する前に	getting_started_sdk.pdf	概要およびサンプルのユースケースが記載されています。
レポートデータベース	report_db.pdf	レポートデータベースの使用に関する情報が含まれています。
ユーザーガイド	user_guide.pdf	Black DuckのUI使用に関する情報が含まれています。

KubernetesまたはOpenShiftの環境にBlack Duckソフトウェアをインストールするには、Helmを使用します。次のリンクをクリックすると、マニュアルが表示されます。

- ・ [Helm](#)は、Black Duckのインストールに使用できるKubernetesのパッケージ マネージャです。Black Duck は Helm3をサポートしており、Kubernetesの最小バージョンは1.13です。

Black Duck 統合に関するドキュメントは、次のリンクから入手できます：

- ・ <https://sig-product-docs.blackduck.com/bundle/detect/page/integrations/integrations.html>
- ・ https://documentation.blackduck.com/category/cicd_integrations

カスタマサポート

ソフトウェアまたはマニュアルについて問題がある場合は、次の Black Duck カスタマー サポートに問い合わせてください。

- ・ オンライン: <https://community.blackduck.com/s/contactsupport>
- ・ サポート ケースを開くには、Black Duck コミュニティ サイト (<https://community.blackduck.com/s/contactsupport>) にログインしてください。
- ・ 常時対応している便利なリソースとして、[オンライン コミュニティ ポータル](#)を利用できます。

Black Duck コミュニティ

Black Duck コミュニティは、カスタマー サポート、ソリューション、情報を提供する主要なオンライン リソースです。コミュニティでは、サポート ケースをすばやく簡単に開いて進捗状況を監視したり、重要な製品情報を確認したり、ナレッジベースを検索したり、他の Black Duck のお客様から情報を得ることができます。コミュニティセンターには、共同作業に関する次の機能があります。

- ・ つながる – サポートケースを開いて進行状況を監視するとともに、エンジニアリング担当や製品管理担当の支援が必要になる問題を監視します。
- ・ 学ぶ – 他の Black Duck 製品ユーザーの知見とベスト プラクティスを通じて、業界をリードするさまざまな企業から貴重な教訓を学ぶことができます。さらに、Customer Hubでは、Black Duckからの最新の製品ニュースやアップデートをいつでもご覧いただけます。これは、当社製品やサービスをより有効に活用し、オープン ソースの価値を組織内で最大限に高めることができます。
- ・ 解決する – Black Duck の専門家や Knowledgebase が提供する豊富なコンテンツや製品知識にアクセスして、探している回答をすばやく簡単に得ることができます。
- ・ 共有する – Black Duckのスタッフや他のお客様とのコラボレーションを通じて、クラウドソースソリューションに接続し、製品の方向性について考えを共有できます。

[Customer Successコミュニティにアクセスしましょう](#)。アカウントをお持ちでない場合や、システムへのアクセスに問題がある場合は、[こちら](#)をクリックして開始するか、community.manager@blackduck.com にメールを送信してください。

トレーニング


Black Duck Customer Education は、Black Duck の教育ニーズをすべて満たすワンストップ リソースです。ここでは、オンライントレーニングコースやハウツービデオへの24時間365日のアクセスを利用できます。

新しいビデオやコースが毎月追加されます。

Black Duck Education では、次を行えます。

- ・ 自分のペースで学習する。
- ・ 希望する頻度でコースを復習する。
- ・ 試験を受けて自分のスキルをテストする。
- ・ 終了証明書を印刷して、成績を示す。

詳細については、<https://blackduck.skilljar.com/page/black-duck> を確認してください。また、Black Duck に関する

ヘルプについては、ヘルプ メニューの [Black Duck チュートリアル]()(Black Duck UIに表示)を選択してください。

Black Duck 包括性と多様性に関する声明

Black Duck は、すべての従業員、お客様、パートナー様が歓迎されていると感じられる包括的な環境の構築に取り組んでいます。当社では、製品およびお客様向けのサポート資料から排他的な言葉を確認して削除しています。また、当社の取り組みには、設計および作業環境から偏見のある言葉を取り除く社内イニシアチブも含まれ、これはソフトウェアやIPに組み込まれている言葉も対象になっています。同時に、当社は、能力の異なるさまざまな人々が当社のWebコンテンツおよびソフトウェアアプリケーションを利用できるように取り組んでいます。なお、当社のIPは、排他的な言葉を削除するための現在検討中である業界標準仕様を実装しているため、当社のソフトウェアまたはドキュメントには、非包括的な言葉の例がまだ見つかる場合があります。

Black Duck セキュリティへの取り組み

Black Duckは、お客様のアプリケーションの保護とセキュリティの確保に専念する組織として、お客様のデータ セキュリティとプライバシーにも同様に取り組んでいます。この声明は、Black Duckのお客様と将来のお客様に、当社のシステム、コンプライアンス認証、プロセス、その他のセキュリティ関連活動に関する最新情報をお届けすることを目的としています。

この声明は次の場所で入手できます。[セキュリティへの取り組み | Black Duck](#)

1. 概要

このドキュメントでは、Docker環境にBlack Duckをインストールする手順について説明します。

Black Duck アーキテクチャ

Black Duck は、Dockerコンテナのセットとして導入されます。さまざまなコンポーネントがコンテナ化されるようにBlack DuckをDocker化することで、Swarmなどのサードパーティのオーケストレーションツールで個々のコンテナをすべて管理できるようになります。

Dockerアーキテクチャは、次のような大幅な改善をBlack Duckにもたらしめます。

- ・ パフォーマンスの向上
- ・ より簡単なインストールと更新
- ・ スケーラビリティ
- ・ 製品コンポーネントのオーケストレーションと安定性

Black Duckアプリケーションを構成するDockerコンテナの詳細については、「[Dockerコンテナ](#)」を参照してください。

Dockerの詳細については、Docker Webサイト(<https://www.docker.com/>)を参照してください。

Dockerのインストール情報を取得するには、<https://docs.docker.com/engine/installation/>にアクセスしてください。

Black Duckサーバーでホストされるコンポーネント

次のリモートBlack Duckサービスは、Black Duckによって使用されます。

- ・ 登録サーバー: Black Duckのライセンスを検証するのに使用されます。
- ・ Black Duck KnowledgeBaseサーバー: Black Duck KnowledgeBase(KB)は、業界で最も包括的な、オープンソース プロジェクト、ライセンス、セキュリティ情報のデータベースです。クラウドのBlack Duck KBを使用することで、Black Duckは、Black Duckインストールを定期的に更新することなく、オープンソースソフトウェア(OSS)に関する最新情報を表示できます。

2. インストールの計画

この章では、Black Duckをインストールする前に実行する必要があるインストール前の計画と構成について説明します。

使用する前に

Black Duckのインストールのプロセスは、Black Duckを初めてインストールするか、以前のバージョンのBlack Duckからアップグレードするか（AppMgrアーキテクチャに基づくか、Dockerアーキテクチャに基づくか）によって異なります。

新規インストール

Black Duckの新規インストールの場合は、次の手順を実行します。

1. この計画に関する章を読んで、すべての要件を確認します。
2. すべての要件を満たしていることを確認したら、インストール手順について[インストール: Black Duck](#)を確認します。
3. [管理タスク] セクションを確認します。

以前のバージョンのからのアップグレード: Black Duck

1. この計画に関する章を読んで、すべての要件を確認します。
2. すべての要件を満たしていることを確認したら、アップグレード手順について[アップグレード Black Duck](#)を確認します。
3. [管理タスク] セクションを確認します。

ハードウェア要件

Black Duck ハードウェアのスケーリングガイドライン

スケーラビリティのサイジングに関するガイドラインについては、「[Black Duck ハードウェアのスケーリング ガイドライン](#)」を参照してください。

Black Duck データベース

⚠ 危険：Black Duckのテクニカルサポート担当者から指示がない限り、Black Duckデータベース（bds_hub）からデータを削除しないでください。必ず適切なバックアップ手順に従ってください。データを削除すると、UIの問題からBlack Duckが完全に起動しなくなるという障害に至る、いくつかのエラーが発生する可能性があります。Black Duck テクニカルサポートは、削除されたデータを再作成することはできません。利用可能なバックアップがない場合、Black Duckは可能な範囲で最善のサポートを提供します。

ディスク容量の要件

必要なディスク容量は、管理するプロジェクトの数によって異なります。したがって、個々の要件が異なる場合があります。各プロジェクトには約200 MBが必要であることを考慮してください。

2. インストールの計画・Dockerの要件

Black Duck Softwareでは、Black Duckサーバーのディスク使用率を監視して、ディスクが最大容量に達しないようにすることを推奨しています。最大容量に達すると、Black Duckで問題が発生する可能性があります。

BDBAのスケーリング


BDBAのスケーリングは、1時間あたりに実行される予想バイナリスキャン数に基づいて、binaryscannerレプリカの数調整し、PostgreSQLリソースを追加することによって行われます。1時間あたり15回のバイナリスキャンごとに、次を追加します。

- ・ 1つのbinaryscannerレプリカ
- ・ PostgreSQL用の1つのCPU
- ・ PostgreSQL用の4 GBのメモリ

予想されるスキャンレートが15の倍数でない場合は、切り上げます。たとえば、1時間あたり24回のバイナリスキャンでは、次のものがが必要です。


- ・ 2つのbinaryscannerレプリカ
- ・ PostgreSQL用の2つの追加CPU、および
- ・ PostgreSQL用の8 GBの追加メモリ。

このガイダンスは、バイナリスキャンが合計スキャンボリューム（スキャン数）の20%以下である場合に有効です。


 注： Black Duck Alertをインストールするには、1 GBの追加メモリが必要です。

Dockerの要件

Black Duckをインストールするための推奨される方法であるDocker Swarmは、Dockerコンテナのクラスタリングおよびスケジューリングツールです。Docker Swarmを使用すると、Dockerノードのクラスタを1つの仮想システムとして管理できます。

 注： スケーラビリティの観点から、Black Duck Softwareでは、単一ノードのSwarm導入でBlack Duckを実行することを推奨しています。スケーラビリティのサイジングの詳細なガイドラインについては、『Black Duckリリース ノート』の「コンテナのスケーラビリティ」セクションを参照してください。

Docker SwarmでBlack Duckを使用する場合、次の制限があります。

 注意： PostgreSQLデータディレクトリでウイルス対策スキャンを実行しないでください。ウイルス対策ソフトウェアは、大量のファイルを開いたり、ファイルをロックしたりします。これらはPostgreSQLの操作を妨げます。特定のエラーは製品によって異なりますが、通常、PostgreSQLがデータファイルにアクセスできなくなります。たとえば、PostgreSQLが「システムで開かれているファイルが多すぎます」というエラーを伴って失敗することがあります。

- ・ データが失われないように、PostgreSQLデータベースは常にクラスタ内の同じノードで実行する必要があります（blackduck-databaseサービス）。

これは、外部PostgreSQLインスタンスを使用したインストールには該当しません。

- ・ blackhack-webappサービスとblackhack-logstashサービスは、同じホスト上で実行する必要があります。

これは、ダウンロードする必要があるログにblackduck-webappサービスがアクセスできるようにするのに必要です。


- ・ 登録データが失われないように、blackduck-registrationサービスは常にクラスタ内の同じノードで実行するか、NFSボリュームまたは同様のシステムによってバックアップされる必要があります。

blackduck-databaseサービスまたはblackduck-webappサービスに使用されるノードと同じノードである必要はありません。

- ・ データが失われないように、blackduck-upload-cacheサービスは常にクラスタ内の同じノードで実行するか、NFSボリュームまたは同様のシステムによってバックアップされる必要があります。
- 他のサービスに使用されるノードと同じノードである必要はありません。

Dockerバージョン

Black Duck のインストールでは、Docker バージョン 23.x、25.0.2 がサポートされています。

 注： Dockerバージョン18.09.x、19.03.x、20.10.xは、Black Duck 2023.7.1 以降サポートされなくなりました。

RHELへのDockerエンジンのインストール


現在Dockerは、s390x (IBM Z) 上のRHEL用のパッケージのみを提供しています。他のアーキテクチャはRHELではまだサポートされていません。詳細については、[Docker](#)と[Redhat](#)のページを参照してください。

オペレーティングシステム

Docker環境にBlack Duckをインストールするのに推奨されるオペレーティングシステムは次のとおりです。

- ・ Red Hat Enterprise Linuxサーバー 8.9および9.x
- ・ Ubuntu 20.04.x
- ・ SUSE Linux Enterprise Serverバージョン12.x(64ビット)
- ・ Oracle Enterprise Linux 7.9

さらに、Black Duckは、サポートされているDockerバージョンに対応している他のLinuxオペレーティング システムをサポートしています。

 注： Docker CEは、Red Hat Enterprise Linux、Oracle Linux、またはSUSE Linux Enterprise Server (SLES) をサポートしていません。詳細については、[ここ](#)をクリックしてください。


Windowsオペレーティングシステムは現在サポートされていません。

ソフトウェア要件

Black Duck は、HTMLインターフェイスを備えたWebアプリケーションです。アプリケーションにはWebブラウザを介してアクセスします。Black Duckでは、次のWebブラウザバージョンがテストされています。

- ・ Safariバージョン17.4.1
 - ・ Safariバージョン14以前はサポートされなくなりました
- ・ Chrome バージョン 123.0.6312.124(公式ビルド)(x86_64)
 - ・ Chromeバージョン91以前はサポートされなくなりました
- ・ Firefox バージョン 124.0.2(64 ビット)
 - ・ Firefoxバージョン89以前はサポートされなくなりました
- ・ Microsoft Edge バージョン 123.0.2420.97(公式ビルド)(64 ビット)
 - ・ Microsoft Edgeバージョン91以前はサポートされなくなりました

Black Duckは互換モードをサポートしていません。

-  注：これらのブラウザバージョンは、Black Duck SoftwareがBlack Duckをテストした現在リリースされているバージョンです。Black Duckのリリース後に新しいブラウザバージョンが利用可能になる場合があります、期待どおりに機能する場合と機能しない場合があります。古いバージョンのブラウザも期待どおりに機能する可能性があります。ただし、テストされておらず、サポートされていない可能性があります。


ネットワーク要件

Black Duck では、次のポートが外部からアクセスできる必要があります。


- ・ ポート443 – NGiNXを介したBlack DuckのWebサーバーHTTPSポート
- ・ ポート55436 – PostgreSQLからの読み取り専用データベースポート(レポート用)

企業のセキュリティポリシーで特定のURLの登録が必要な場合、Black DuckインストールからBlack Duck Softwareホストサーバーへの接続は、ポート443でのHTTPS/TCPを介した次のサーバーとの通信に制限されます。

- ・ updates.suite.blackducksoftware.com (ソフトウェア登録用)
- ・ kb.blackducksoftware.com (Black Duck KBデータへのアクセス)
- ・ https://auth.docker.io/token?scope=repository/blackducksoftware/blackduckregistration/pull&service=registry.docker.io (Dockerレジストリへのアクセス)
- ・ data.reversinglabs.comとapi.reversinglabs.com (ReversingLabsスキャンが有効な場合)

-  注：ネットワークプロキシを使用している場合は、これらのURLをプロキシ構成で宛先として設定する必要があります。

許可リストのアドレスと IP 範囲

-  注：HTTPS は Black Duck へのすべてのトラフィックに使用されます。サブネット マスクを含む IP (たとえば、103.21.244.0/22 の「/22」部分) は IP の範囲を表しており、Black Duck を意図したとおりに機能させるためには、そのすべてを許可リストに登録する必要があります。


許可リストに次のアドレスと IP があることを確認します。

ドメイン	IP アドレス
kb.blackducksoftware.com	34.160.126.173、34.149.112.69、34.111.46.24、35.224.73.200、35.242.23
updates.suite.blackducksoftware.com	35244241173
scass.blackduck.com	35.244.200.22
repo.blackduck.com	34.149.5.115
production.cloudflare.docker.com	173.245.48.0/20、103.21.244.0/22、103.22.200.0/22、103.31.4.0/22、14
hub.docker.com	44.219.3.189、3.224.227.198、44.193.181.103
docker.io	44.219.3.189、3.224.227.198、44.193.181.103
auth.docker.io	34.226.69.105、54.196.99.49、3.219.239.5
registry-1.docker.io	54.196.99.49、3.219.239.5、34.226.69.105
github.com	140.82.116.4
data.reversinglabs.com	104.18.24.126、104.18.25.126
api.reversinglabs.com	185.64.132.12

接続性の検証

接続を確認するには、次の例に示すようにcURLコマンドを使用します。

```
curl -v https://kb.blackducksoftware.com
```


 ヒント： Dockerホストで接続を確認することもできますが、Dockerネットワーク内から接続を確認することをお勧めします。

IPv4およびIPv6ネットワーク


Black Duck は、入出力トラフィックの両方でIPv4とIPv6をサポートします。ただし、内部Black Duckコンテナ ネットワークが正しく機能するには、IPv4が必要です。具体的には、Black DuckはBlack Duckコンテナ クラスタからNGiNXへのネットワークトラフィックのイン/アウトバウンドのIPv6を処理できますが、クラスタ内の内部トラフィックはIPv4を使用する必要があります。

データベース要件

Black Duck では、PostgreSQLオブジェクトリレーショナルデータベースを使用してデータを格納します。

 注意： Black Duckのテクニカル サポート担当者から指示がない限り、Black Duckデータベース(bds_hub)からデータを削除しないでください。必ず適切なバックアップ手順に従ってください。データを削除すると、UIの問題からBlack Duckが完全に起動しなくなるという障害に至る、いくつかのエラーが発生する可能性があります。Black Duck テクニカルサポートは、削除されたデータを再作成することはできません。利用可能なバックアップがない場合、Black Duckは可能な範囲で最善のサポートを提供します。


Black Duckをインストールする前に、自動的にインストールされるデータベースコンテナを使用するか、外部のPostgreSQLインスタンスを使用するかを判断します。

 重要： Black Duck 2024.10.0 の時点で、Black Duck では、外部 PostgreSQL を使用する新規インストールには PostgreSQL 16.x を使用することを推奨しています。PostgreSQL 14.x は、外部 PostgreSQL インスタンスではサポートされなくなりました。内部 PostgreSQL コンテナを使用している場合、PostgreSQL 15 は引き続きサポート対象バージョンです。

外部PostgreSQLインスタンスの場合、Black Duckは以下をサポートしています。

- ・ Amazon Relational Database Service (RDS)を介したPostgreSQL 15.x、16.x
- ・ Google Cloud SQL を介した PostgreSQL 15.x、16.x
- ・ PostgreSQL 15.x、16.x (Community Edition)
- ・ Microsoft Azure を介した PostgreSQL 15.x、16.x

詳細については、「[外部PostgreSQLインスタンスの構成](#)」を参照してください。

 注： PostgreSQL のサイジングに関するガイドラインについては、「[Black Duck ハードウェアのスケーリングガイドライン](#)」を参照してください。


PostgreSQLのバージョン

Black Duck 2023.10.0では、新しいPostgreSQLの機能がサポートされており、Black Duckサービスのパフォーマンスと信頼性が向上します。Black Duck 2023.10.0の時点では、内部PostgreSQLコンテナ用にサポートされているPostgreSQLのバージョンはPostgreSQL 14です。


Black Duck 2023.10.0以降、PostgreSQLコンテナを使用する導入では、PostgreSQLの設定は自動で設定されます。外部PostgreSQLを使用するお客様は、設定を引き続き手動で適用する必要があります。


2. インストールの計画・Swarmでの移行の概要

PostgreSQLコンテナを使用していてBlack Duckのバージョン2022.2.0～2023.7.x(表記を含む)からアップグレードするお客様は、PostgreSQL 14へ自動で移行されます。さらに古いバージョンのBlack Duckからアップグレードするお客様は、2024.7.0へアップグレードする前に、2023.7.xへアップグレードする必要があります。

 注：PostgreSQL のサイジングに関するガイドラインについては、「[Black Duck ハードウェアのスケーリングガイドライン](#)」を参照してください。

独自の外部PostgreSQLインスタンスを実行する場合、Black Duckは、新規インストールに最新バージョンのPostgreSQL 16を使用することをお勧めします。

 注：Black Duck 2024.4.0では、テスト目的でのみPostgreSQL 16を外部データベースとして使用するための事前サポートが追加されました。Black Duck 2024.7.0以降では、PostgreSQL 16は本番環境での使用に完全対応しています。

 注意：PostgreSQLデータディレクトリでウイルス対策スキャンを実行しないでください。ウイルス対策ソフトウェアは、大量のファイルを開いたり、ファイルをロックしたりします。これらはPostgreSQLの操作を妨げます。特定のエラーは製品によって異なりますが、通常、PostgreSQLがデータファイルにアクセスできなくなります。たとえば、PostgreSQLが「システムで開かれているファイルが多すぎます」というエラーを伴って失敗することがあります。

一般的な移行プロセス

このガイドンスは、任意のPG 9.6ベースのHub(2022.2.0より前のリリース)から2022.10.0以降にアップグレードする場合に該当します。

1. 移行は、blackadue-postgres-upgraderコンテナによって実行されます。
2. PostgreSQL 9.6ベースのBlack Duckバージョンからアップグレードする場合：
 - ・ 将来のPostgreSQLバージョンのアップグレードがより簡単になるように、PostgreSQLデータボリュームのフォルダレイアウトが再構成されます。
 - ・ データボリュームの所有者のUIDが変更されます。新しいデフォルトUIDは1001です。ただし、導入固有の説明を参照してください。
3. pg_upgradeスクリプトを実行して、データベースをPostgreSQL 13に移行します。
4. クエリプランナ統計情報を初期化するために、PostgreSQL 13データベース上でプレーンなANALYZEが実行されます。
5. blackduck-postgres-upgraderが終了します。

Swarmでの移行の概要

- ・ 移行は完全に自動化されているため、Black Duckの標準アップグレードの操作以外に追加の操作は不要です。
- ・ 上記のレイアウトとUIDの変更を行うには、blackduck-postgres-upgraderコンテナをルートとして実行する必要があります。
- ・ その後のBlack Duckの再起動時に、blackadu-postgres-upgraderは移行が不要であると判断し、すぐに終了します。
- ・ オプション: 移行が成功した後は、blackduck-postgres-upgraderコンテナをルートとして実行する必要はありません。

4.2.0より前のバージョンのBlack Duckからアップグレードする場合は、データベースの移行手順について、第6章「Black Duckのアップグレード」を参照してください。

プロキシサーバーの要件

Black Duck は、次のものをサポートしています。


- ・ 認証なし
- ・ Digest
- ・ Basic
- ・ NTLM

Black Duckにプロキシリクエストを行う場合は、プロキシサーバー管理者と協力して、次の必要な情報を取得します。

- ・ プロキシサーバーホストで使用されるプロトコル(httpまたはhttps)。
- ・ プロキシサーバーホストの名前
- ・ プロキシサーバーホストがリスンするポート。

連携するNGINXサーバーの設定：Black Duck

Black Duckの前にHTTPSサーバー/プロキシとして機能するNGINXサーバーがある場合は、NGINXサーバーが正しいヘッダーをBlack Duckに渡すように、NGINX構成ファイルを変更する必要があります。Black Duck により、HTTPSを使用するURLが生成されます。

 注：NGINXサーバー上の1つのサービスのみがhttpsポート443を使用できます。


Black Duckに正しいヘッダーを渡すには、nginx.config構成ファイルのlocationブロックを次のように編集します。

```
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;
}
```

プロキシ サーバー/ロード バランサー構成でX-Forwarded-Prefixヘッダーが指定されている場合は、nginx.conf構成ファイルのlocationブロックを次のように編集します。

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

ファイルを正常にスキャンするには、context/パラメータを使用するか(コマンドライン使用時)、Black Duck Scannerの[Black DuckサーバーURL]フィールドにパラメータを含める必要があります。

 注：これらの手順はNGINXサーバーに該当しますが、どのタイプのプロキシサーバーでも同様の構成変更を行う必要があります。

プロキシサーバーがBlack Duckへのリクエストを書き換える場合は、次のHTTPヘッダーを使用して元のリクエスト元ホストの詳細を保持できることをプロキシサーバー管理者に知らせてください。

HTTPヘッダー	説明
X-Forwarded-Host	リクエストを行うために書き換えられたまたはルーティングされたホストのリストを追跡します。元のホストは、カンマ区切りリストの最初のホストです。 例:

	X-Forwarded-Host: "10.20.30.40,my.example, 10.1.20.20"
X-Forwarded-Port	元のリクエストに使用されたポートを表す1つの値が含まれます。 例: X-Forwarded-Port: "9876"
X-Forwarded-Proto	元のリクエストに使用されたプロトコルスキームを表す1つの値が含まれます。 例: X-Forwarded-Proto: "https"
X-Forwarded-Prefix	元のリクエストに使用されたプレフィックスパスが含まれます。 例: X-Forwarded-Prefix: "prefixPath" ファイルを正常にスキャンするには、contextパラメータを使用する必要があります

Amazonサービス

次の操作を実行できます。

- Black DuckをAmazon Web Services (AWS)にインストールする
AWSの詳細については、[AWSドキュメント](#)と[AMIドキュメント](#)を参照してください。
 - Black Duckで使用するPostgreSQLデータベースにAmazon Relational Database Service (RDS)を使用する。
Amazon RDSの詳細については、[Amazon Relational Database Serviceドキュメント](#)を参照してください。
- !** 重要: Black Duck では、PostgreSQL 15 (外部PostgreSQLデータベース)を使用することを推奨しています。
外部PostgreSQLインスタンスでは、数字のみで構成されるユーザー名がサポートされるようになりました。

ポートに関するその他の情報

次のリストのポートは、ファイアウォールルールまたはDocker構成でブロックすることはできません。これらのポートがどのようにブロックされるかの例を次に示します。

- ホストマシン上のiptables構成。
- ホストマシン上のfirewalld構成。
- ネットワーク上の別のルータ/サーバーでの外部ファイアウォール構成。
- Dockerがデフォルトで作成するもの、およびBlack Duckがデフォルトで作成するもの以外の適用される特別なDockerネットワークルール。

ブロック解除されたままにする必要があるポートの完全なリストは次のとおりです。

- 443
- 8443
- 8000
- 8888
- 8983
- 16543

- ・ 17543
- ・ 16545
- ・ 16544
- ・ 55436

キープアライブ設定の構成

net.ipv4.tcp_keepalive_time/パラメータは、確立されているTCP接続をアプリケーションがアイドル状態にしておく時間を制御します。デフォルトでは、この値は7,200秒(2時間)です。

最適なBlack Duckパフォーマンスを実現するには、このパラメータの値を600～800秒にする必要があります。

この設定は、Black Duckのインストール前またはインストール後に設定できます。

値を編集するには、次の手順を実行します。

1. /etc/sysctl.confファイルを編集します。以下に例を示します。

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

sysctlコマンドを使用してこのファイルを変更することもできます。
2. net.ipv4.tcp_keepalive_timeを追加するか(パラメータがファイルにない場合)、既存の値を編集します(パラメータがファイルにある場合)。

```
net.ipv4.tcp_keepalive_time = <value>
```
3. ファイルを保存して終了します。
4. 次のコマンドを入力して、新しい設定を読み込みます。

```
sysctl -p
```
5. Black Duckがインストールされている場合は、それを再起動します。

ナレッジベースフィードバックサービス

ナレッジベース フィードバックは、Black Duck KnowledgeBase(KB)機能を強化するのに使用されます。

- ・ KBによって実行されたマッチのコンポーネント、バージョン、取得元、取得元ID、またはライセンスに構成表の調整が加えられた場合、フィードバックが送信されます。
- ・ コンポーネントにマッチしないファイルが特定された場合にも、フィードバックが送信されます。手動で追加されたコンポーネントにファイルが関連付けられていない場合、フィードバックは送信されません。
- ・ フィードバックは、将来のマッチの精度を向上させるのに使用されます。この情報は、Black Duckがリソースに優先順位を付けて、お客様にとって重要なコンポーネントをより詳細に検査できるようにするのに役立ちます。

! 重要： お客様を特定する情報はKBに送信されません。

ユーザーエージェント解析


ナレッジベースは、発信元ユーザーエージェント解析を使用して、ナレッジベースサービスのスケーラビリティを向上させ、ユーザーに対するサービス品質を向上させます。

追加のヘッダー情報により、ナレッジベースへの送信HTTPリクエストのヘッダーサイズが増加します。追加のヘッダーサイズをサポートするために、一部の中間エグレスプロキシ(顧客管理)で再構成が必要になる場合がありますが、このようになることはほとんどありません。

フィードバックサービスの無効化

ナレッジベースフィードバックサービスはデフォルトで有効です。構成表に対して実行された調整はナレッジベースに送信されます。

- ・ `BLACKDUCK_KBFEEDBACK_ENABLED`環境変数を使用して、フィードバックサービスを上書きできます。`false`の値を指定すると、フィードバックサービスが上書きされ、構成表調整はナレッジベースに送信されません。
- ・ フィードバックサービスを無効にするには、`blackduck-config.env` fileに`BLACKDUCK_KBFEEDBACK_ENABLED=false`を追加します。

 注：フィードバックサービスを再度有効にするには、値を[真]に設定します。

3. インストール: Black Duck

Black Duckをインストールする前に、次の要件を満たしていることを確認してください。

Black Duck インストール要件	
ハードウェア要件	
√	ハードウェアが最小 ハードウェア要件 を満たしていることを確認しました。
Dockerの要件	
√	システムが dockerの要件 を満たしていることを確認しました。
ソフトウェア要件	
√	システムと潜在的なクライアントが ソフトウェア要件 を満たしていることを確認しました。
ネットワーク要件	
√	ネットワークが ネットワーク要件 を満たしていることを確認しました。具体的な内容: <ul style="list-style-type: none"> ・ポート443とポート55436は外部からアクセス可能です。 ・サーバーは、Black Duckライセンスの検証に使用されるupdates.suite.blackducksoftware.comへのアクセス権を有しています。
データベース要件	
√	データベース構成 を選択しました。 具体的な内容: データベース設定を構成 しました(外部PostgreSQLインスタンスを使用している場合)。
プロキシ要件	
√	ネットワークが プロキシ要件 を満たしていることを確認しました。 Black Duckのインストール前またはインストール後に プロキシ設定 を構成します。
Webサーバーの要件	
√	Black Duckのインストール前またはインストール後に Webサーバーの設定 を構成します。

インストールファイル

インストールファイルはGitHubで入手できます。

オーケストレーションファイルをダウンロードします。インストール/アップグレードプロセスの一環として、これらのオーケストレーションファイルは必要なDockerイメージをプルダウンします。

tar.gzファイルのファイル名はアクセス方法によって異なりますが、内容は同じです。

GitHubページからダウンロードする場合

1. リンクを選択して、GitHubページからtar.gzファイルをダウンロードします: <https://github.com/blackducksoftware/hub>。
2. Black Duck .gzファイルを解凍します。
`gunzip hub-2024.10.0.tar.gz`

3. インストール: Black Duck・配布

3. Black Duck.tarファイルを展開します。

```
tar xvf hub-2024.10.0.tar
```

wgetコマンドを使用してダウンロードする場合

1. 次のコマンドを実行します。

```
wget https://github.com/blackducksoftware/hub/archive/v2024.10.0.tar.gz
```

2. Black Duck .gzファイルを解凍します。

```
gunzip v2024.10.0.tar.gz
```

3. Black Duck.tarファイルを展開します。

```
tar xvf v2024.10.0.tar
```

配布

docker-swarmディレクトリは、Black Duckをインストールまたはアップグレードするのに必要な次のファイルで構成されています。

- blackduck-config.env: Black Duckの設定を構成するための環境ファイル。
- docker-compose.bdba.yml: Black DuckをBlack Duck — Binary Analysisとともにインストールし、Black Duckによって提供されたデータベース コンテナを使用する場合に使用されるDocker Composeファイル。
- docker-compose.dbmigrate.yml: Black Duckによって提供されたデータベースコンテナを使用する場合に、PostgreSQLデータベースを移行するのに使用されるDocker Composeファイル。
- docker-compose.externaldb.yml: 外部PostgreSQLデータベースで使われるDocker Composeファイル。
- docker-compose.local-overrides.yml: .ymlファイルのデフォルト設定を上書きするのに使用されるDocker Composeファイル。
- docker-compose.readonly.yml: Swarmサービスに対してファイルシステムを読み取り専用として宣言するDocker Composeファイル。
- docker-compose.yml: Black Duckによって提供されたデータベースコンテナを使用する場合のDocker Composeファイル。
- external-postgres-init.pgsql: 外部PostgreSQLデータベースの構成に使用されるPostgreSQL.sqlファイル。
- hub-bdba.env: Black Duck Binary Analysisの追加設定が含まれている環境ファイル。このファイルを変更する必要はありません。
- hub-postgres.env: [外部PostgreSQLデータベース](#)を設定するための環境ファイル。
- hub-webserver.env: [Webサーバー設定を構成](#)するための環境ファイル。

binディレクトリは、次のファイルで構成されています。

- bd_get_source_upload_master_key.sh: [ソースファイルのアップロード](#)時にマスターとシールキーをバックアップするのに使用されるスクリプト。
- hub_create_data_dump.sh: Black Duckによって提供されたデータベースコンテナを使用する場合に、PostgreSQLデータベースをバックアップするのに使用されるスクリプト。
- hub_db_migrate.sh: Black Duckによって提供されたデータベースコンテナを使用する場合に、PostgreSQLデータベースを移行するのに使用されるスクリプト。
- hub_reportdb_changepassword.sh: [レポートデータベースのパスワードを変更](#)および設定するのに使用されるスクリプト。

- ・ `recover_master_key.sh`: [ソースファイルのアップロード](#)に使用する新しいシールキーを作成するスクリプト。
- ・ `system_check.sh`: [Black Duckシステム情報を収集](#)してカスタマサポートに送信するのに使用されるスクリプト。

`sizes-gen02`ディレクトリは、次のファイルで構成されています。

- ・ `resources.yaml`: これらは、拡張スキャン用のリソース定義ファイルです。

`sizes-gen03`ディレクトリは、次のファイルで構成されています。


- ・ `10sph.yaml`, `120sph.yaml`, `250sph.yaml`, `500sph.yaml`, `1000sph.yaml`, `1500sph.yaml`, `2000sph.yaml`: これらは、さまざまなスキャン/時間サイズに関するリソース定義ファイルです。詳細については、「[ハードウェア要件](#)」セクションを参照してください。

`sizes-gen04`ディレクトリは、次のファイルで構成されています。

- ・ `10sph.yaml`, `120sph.yaml`, `250sph.yaml`, `500sph.yaml`, `1000sph.yaml`, `1500sph.yaml`, `2000sph.yaml`: これらは、Black Duck 2024.1.0で導入されたさまざまなスキャン/時間サイズに関するリソース定義ファイルです。詳細については、「[ハードウェア要件](#)」セクションを参照してください。

インストール: Black Duck

Black Duckをインストールする前に、構成する必要がある設定があるかどうかを確認してください。システム設定の詳細については、「[管理タスク](#)」セクションを参照してください。

 注: これらの手順は、Black Duckの新規インストール用です。[Black Duckのアップグレード](#)の詳細については、第6章を参照してください。

次のBlack Duckのインストール手順では、`docker`グループのユーザーまたはルートユーザーであるか、`sudo`アクセス権を持っている必要があります。ルート以外のユーザーとしてBlack Duckをインストールするには、次のセクションを参照してください。

PostgreSQLデータベースコンテナを使用したBlack Duckのインストール

1.

```
docker swarm init
```

`docker swarm init`コマンドは、単一ノードのswarmを作成します。

2.

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-overrides.yml hub
```

Black Duck 2022.4.0以降では、コンテナ リソースを`docker-compose.yml`に直接は割り当てなくなりました。代わりに、リソースは別の上書きファイルで指定されます。Black Duck 2022.4.0より前に使用されていたリソース割り当ては、`sizes-gen02/resources.yaml`にあります。Black Duck 2024.1.0以降では、`sizes-gen04`フォルダで複数の割り当てが可能です。1時間あたりの平均スキャン数で測定された負荷に基づいて、**7つの割り当て**があります。予想される負荷が事前定義された割り当てのいずれにも一致しない場合は切り上げます。

上記の例では、

- ・ `docker-compose.yml`: ストック導入。このファイルは編集してはいけません。
- ・ `sizes-gen04/120.yaml`: リソース定義ファイル。この例では、`sizes-gen04/120.yaml`が目的のリソース定義として使用されていますが、これはスキャンパターンと使用法に合わせて変更できます。使用可能なすべてのオプションのリストについては、[配布](#)セクションを参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。
- ・ `docker-compose.local-overrides.yml`: インストール環境にシークレット、メモリ制限などのカスタム構成がある場合のオプションファイル。

PostgreSQLデータベースコンテナを使用したBlack DuckとBlack Duck Binary Analysisのインストール

1.

```
docker swarm init
```

docker swarm initコマンドは、単一ノードのswarmを作成します。

2.

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.bdba.yml hub
```

上記で利用したリソース定義ファイルは、スキャン パターンと使用法に合わせて変更できます。利用可能なすべてのオプションのリストについては、[配布](#)セクションを参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。

外部PostgreSQLインスタンスを使用したBlack Duckのインストール

1.

```
docker swarm init
```

docker swarm initコマンドは、単一ノードのswarmを作成します。

2.

```
docker stack deploy -c docker-compose.externaldb.yml -c sizes-gen04/120sph.yaml hub
```

上記で利用したリソース定義ファイルは、スキャン パターンと使用法に合わせて変更できます。利用可能なすべてのオプションのリストについては、[配布](#)セクションを参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。

外部PostgreSQLインスタンスを使用したBlack DuckとBlack Duck Binary Analysisのインストール

1.


```
docker swarm init
```

docker swarm initコマンドは、単一ノードのswarmを作成します。

2.

```
docker stack deploy -c docker-compose.externaldb.yml -c sizes-gen04/120sph.yaml -c docker-compose.bdba.yml hub
```

上記で利用したリソース定義ファイルは、スキャン パターンと使用法に合わせて変更できます。利用可能なすべてのオプションのリストについては、[配布](#)セクションを参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。

 **注：**新しいガイダンスファイル(導入サイジング用)を外部データベースで使用する場合は、導入前に、使用するTシャツサイズファイルからpostgresおよびpostgres-upgraderサービスを削除します。

ファイルシステムが読み取り専用の状態でのBlack Duckのインストール

1.

```
docker swarm init
```


docker swarm initコマンドは、単一ノードのswarmを作成します。

2.

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.readonly.yml hub
```

上記で利用したリソース定義ファイルは、スキャン パターンと使用法に合わせて変更できます。利用可能なすべてのオプションのリストについては、[配布](#)セクションを参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。

Swarmサービスに対してファイルシステムが読み取り専用の状態でBlack Duckをインストールするには、前の手順にdocker-compose.readonly.ymlファイルを追加します。

 注: Dockerの一部のバージョンでは、イメージがプライベートリポジトリに存在する場合に、上記のコマンドに次のフラグを追加しない限り、dockerスタックはそれらをプルしません。--with-registry-auth.

インストールの確認


docker psコマンドを実行して各コンテナのステータスを表示することで、インストールが成功したことを確認できます。「正常」ステータスは、インストールが正常に完了したことを示します。インストール後、数分間にわたってコンテナが「開始」状態になることがあります。

Black Duckのすべてのコンテナが起動すると、Black DuckのWebアプリケーションがポート443上でDockerホストに公開されます。[ホスト名](#)を構成したことを確認します。これで、次のように入力してBlack Duckにアクセスすることができます。

<https://hub.example.com>

Black Duckに初めてアクセスすると、登録とエンドユーザーライセンス契約が表示されます。Black Duckを使用するには、条件に同意する必要があります。

提供された登録キーを入力して、Black Duckにアクセスします。

 注: 再登録する必要がある場合は、エンドユーザーライセンス契約の条件に再度同意する必要があります。

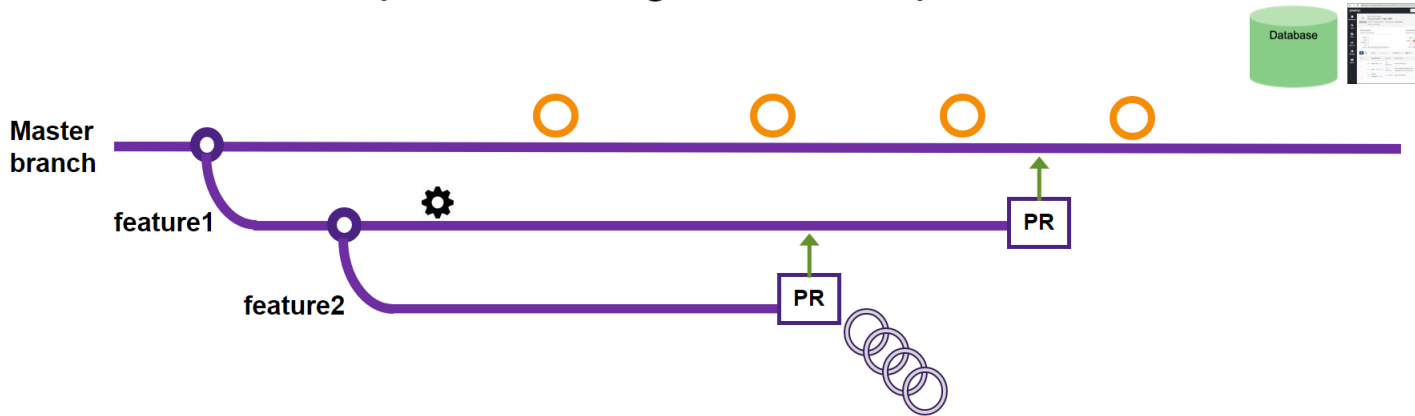
高速スキャン: Black Duck

Black Duck 高速スキャンは、Black Duck 2021.6.0でデフォルトで利用できます。

高速スキャン(依存関係スキャン)は、Black Duckで構成表を作成することなく、マージ前にコードの脆弱性やポリシー違反をチェックする効率的な方法を開発者に提供します。

高速スキャンを使用するには、Black Duck Detect 7.0.0以降を使用する必要があります。

Black Duck rapid scanning for development use cases



- **Rapid Scan**
Early check for vulns or policy violations at code commit / pull request
- **Full SCA Scan**
Complete SCA scan to determine all License & Security Risk

- ✓ Scan and do quick check before code commit
- ✓ Pass/fail indicator based on policy violations
- ✓ Visibility into existing vulnerabilities
- ✓ Quick (<1 min) and scalable (>30k scans/day)

4. 管理タスク

環境ファイルと変数

環境ファイルの使用

一部の構成では環境ファイルを使用します(例: Webサーバー設定、プロキシ設定、または外部PostgreSQL設定の構成)。これらの設定を構成する環境ファイルは、`docker-swarm`ディレクトリにあります。

環境ファイルを使用する設定を構成するには、次の手順を実行します。

- ・ Black Duckをインストールする前に構成設定を行うには、以下の説明に従ってファイルを編集し、変更を保存します。
- ・ Black Duckをインストールした後に既存の設定を変更するには、設定を変更してから、スタック内の[サービスを再導入](#)します。

環境変数とバイナリのスキャン

Black Duck Binary Analysis (BDBA)を使用してバイナリをスキャンする場合は、構成表にバージョンのないコンポーネントが表示されるように、`Job Runner`コンテナ環境変数に`HUB_SCAN_ALLOW_PARTIAL= 'true'`パラメータが追加されていることを確認する必要があります。BDBAスキャナは、Black Duckのスキャンとは異なり、バージョン文字列情報がバイナリで識別できない場合にバージョンのないコンポーネントを表示します。構成表では、コンポーネントの名前の横に疑問符(?)が表示され、コンポーネントにセキュリティ脆弱性を割り当てる前に、このコンポーネントを確認する必要があることが示されます。これは、Black Duckでは、セキュリティ脆弱性をコンポーネントにマッピングするにはバージョンが必要であるためです。


重複している構成表の検出

スキャンパフォーマンスを向上させるため、重複している構成表の検出機能はデフォルトで有効になっています。

この機能により、スキャンによって既存の構成表と同一の構成表が生成されると判断された場合、構成表の計算がスキップされます。無効にするには、次の設定を使用します。

```
SCAN_SERVICE_OPTS=-Dbblackduck.scan.disableRedundantScans=true
```


この設定は、`blackduck-config.env`ファイルで変更できます。

 注: Black Duck 2021.4.0では、`Detect`によって検出された一連の依存関係が前のスキャンのセットと同一の場合にのみ、この機能がパッケージマネージャ(依存関係)スキャンに影響を与えます。この機能は今後のリリースで拡張される予定です。

Bearerトークンの有効期限の変更

REST APIで使用するBearerトークンの有効期限を延長するには、`docker-compose.local-overrides.yml`ファイルを使用して、`HUB_AUTHENTICATION_ACCESS_TOKEN_EXPIRE`環境変数に新しい有効期限値(秒単位)を構成して、デフォルト設定を上書きします。


`HUB_AUTHENTICATION_ACCESS_TOKEN_EXPIRE`プロパティは、アクセストークンの有効期限が切れるまでの秒数です。

 注：有効期限の変更は、docker-compose.local-overrides.ymlファイルで設定を変更した後に作成されたAPIトークンに対してのみ機能します。設定を変更してサービスを再起動したときに、既存のデータベースレコード/APIトークンは設定した有効期限に更新されません。

KBMATCH_SENDFPATHパラメータについて

KBMATCH_SENDFPATH: このパラメータは、ナレッジベースで、マッチング目的および精度でファイルパスとファイル名が使用されないようにします。Black Duck では、マッチング結果に何らかの影響を与える可能性があるため、これを変更することを推奨していません。

Black Duckでパス マッチングをオフにするには、blackduck-config.env内でシステム プロパティKBMATCH_SENDFPATHをfalseに設定します(デフォルトはtrueに設定されます)。

 注：このパラメータは、2023.4.0で廃止されました。今後の更新では削除される予定です。Black Duck のユーザーで、Match as a Service(MaaS)が有効になっているユーザーは、このパラメータを使用できません。このオプションを引き続き使用する場合は、Black Duckサポートに連絡して、Black Duck登録キーに対応しているMaaSを無効にしてもらう必要があります。

プロキシサーバー経由でのAPIドキュメントへのアクセス

リバースプロキシを使用していて、そのリバースプロキシのサブパスの下にBlack Duckがある場合は、APIドキュメントにアクセスできるようにBLACKDUCK_SWAGGER_PROXY_PREFIXプロパティを構成します。BLACKDUCK_SWAGGER_PROXY_PREFIXの値はBlack Duckのパスです。たとえば、「https://customer.companyname.com/hub」にあるBlack Duckにアクセスした場合、BLACKDUCK_SWAGGER_PROXY_PREFIXの値は「hub」になります。

このプロパティを構成するには、docker-swarmディレクトリにあるblackduck-config.envファイルを編集します。

Black Duckサーバー以外からのREST APIへのアクセス

Black Duckサーバー以外から提供されたWebページからBlack DuckREST APIにアクセスしたい場合があります。Black Duckサーバー以外からREST APIにアクセスできるようにするには、Cross Origin Resource Sharing(CORS)を有効にする必要があります。

Black DuckインストールのCORSを有効にして構成するためのプロパティは次のとおりです。

プロパティ	説明
BLACKDUCK_HUB_CORS_ENABLED	必須。CORSを有効にするかどうかを定義します。「true」はCORSが有効であることを示します。
BLACKDUCK_CORS_ALLOWED_ORIGINS_PROP_NAME	必須。CORSの許可されているオリジン。ブラウザは、クロスオリジンリクエストを行うときにオリジンヘッダーを送信します。このオリジンは、blackduck.hub.cors.allowedOrigins/BLACKDUCK_CORS_ALLOWED_ORIGINS_PROP_NAMEプロパティにリストされている必要があります。たとえば、http://123.34.5.67:8080 からページを提供するサーバーを実行している場合は、ブラウザはこれを発信元として設定し、この値をプロパティに追加する必要があります。

プロパティ	説明
	プロトコル、ホスト、およびポートが一致している必要があります。複数のベースオリジンURLを指定するには、カンマ区切りリストを使用します。
BLACKDUCK_CORS_ALLOWED_HEADERS_PROP_NAME	オプション。リクエストの作成に使用できるヘッダー。
BLACKDUCK_CORS_EXPOSED_HEADERS_PROP_NAME	オプション。CORSをリクエストするブラウザがアクセスできるヘッダー。
BLACKDUCK_CORS_ALLOWED_ORIGIN_PATTERNS_PROP_NAME	ワイルドカードパターンで宣言されたオリジンをサポートするBLACKDUCK_CORS_ALLOWED_ORIGINS_PROP_NAME*,*の代替。BLACKDUCK_CORS_ALLOWED_ORIGIN_PATTERNS_PROP_NAMEはBLACKDUCK_CORS_ALLOWED_ORIGINS_PROP_NAMEを上書きします。
BLACKDUCK_CORS_ALLOW_CREDENTIALS_PROP_NAME	ブラウザが、クロスドメインリクエストとともにcookieなどの資格情報をアノテーション付きエンドポイントに送信する必要があるかどうかを指定します。構成された値は、プリフライトリクエストのAccess-Control-Allow-Credentials応答ヘッダーに設定されます。ALLOW_CREDENTIALS=trueおよびALLOWED_ORIGIN=*に構成することは無効です。許可されたオリジンに「ALL」構成値が必要な場合は、今後は代わりにALLOWED_ORIGIN_PATTERNS構成プロパティを使用して構成する必要があります。

これらのプロパティを構成するには、docker-swarmディレクトリにあるblackduck-config.envファイルを編集します。

ダッシュボードの更新間隔の構成

blackduck-config.envファイルでSEARCH_DASHBOARD_REFRESH_JOB_DUTY_CYCLE環境変数を構成して、ダッシュボードの更新間隔をスケジュールします。

許可されているSEARCH_DASHBOARD_REFRESH_JOB_DUTY_CYCLE値は次のとおりです。

- ・ デフォルト値は20です。
- ・ 最小値は10です。
- ・ 最大値は50です。

値が、許可されている値と一致しない場合は、最も近い許可されている値にリセットされます。

例:

SEARCH_DASHBOARD_REFRESH_JOB_DUTY_CYCLE=100 構成された値は、最も近い許可されている値(10、20、または50)である50にリセットされます。

SEARCH_DASHBOARD_REFRESH_JOB_DUTY_CYCLE=10 構成された値は、許可されている値である10のままになります。

SEARCH_DASHBOARD_REFRESH_JOB_DUTY_CYCLE=5 構成された値は、最も近い許可されている値(10、20、または50)である10にリセットされます。

証明書の管理

デフォルトでは、Black DuckはHTTPS接続を使用します。HTTPSの実行に使用されるデフォルトの証明書は自己署名証明書です。これは、ローカルで作成され、承認された認証局(CA)によって署名されていないことを意味します。

このデフォルトの証明書を使用する場合、Black DuckのUIにログインするにはセキュリティ例外を作成する必要があります。ブラウザは証明書の発行元を認識しないため、デフォルトでは受け入れられません。

また、スキャン時にBlack Duckサーバーに接続するときに証明書に関するメッセージを受け取ります。これは、証明書は自己署名であり、CAによって発行されたものではないため、スキャナが証明書を検証できないためです。

目的の認証局から署名付きSSL証明書を取得できます。署名付きSSL証明書を取得するには、証明書署名要求(CSR)を作成します。CAはこれを使用して、Black Duckインスタンスを実行しているサーバーを「安全」として識別する証明書を作成します。署名付きSSL証明書をCAから受信したら、自己署名証明書を置き換えることができます。


SSL証明書キーストアを作成するには、次の手順を実行します。

1. SSLキーとCSRを生成するには、コマンドラインで次のように入力します。

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

ここで、

- ・ <keyfile>は、<会社のサーバー名>.key
- ・ <keystrength>は、サイトの公開暗号化キーのサイズ
- ・ <CSRfile>は、<会社のサーバー名>.csr

 **注：** 会社のサーバーとして入力する名前は、SSLサーバーが存在する完全なホスト名であり、組織名はドメインの「whois」レコードにあるものと同じであることが重要です。

以下に例を示します。

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

この例では、server.company.comのCSRを作成して、CAから証明書を取得します。

2. CSRを希望する方法でCAに送信します(通常はWebポータル経由)。
3. Apache Webサーバーの証明書が必要であることを示します。
4. 会社に関する要求された情報をCAに提供します。この情報は、ドメインレジストリ情報と一致する必要があります。
5. CAから証明書を受け取ったら、次のセクションの手順を実行して、証明書をBlack Duckインスタンスにアップロードします。

カスタム証明書の使用

webserverコンテナには、Dockerから取得した自己署名証明書があります。この証明書をカスタム証明書-キーペアに置き換えることができます。

1. docker secretコマンドでWEBSERVER_CUSTOM_CERT_FILEとWEBSERVER_CUSTOM_KEY_FILEを使用し、Docker Swarmに証明書とキーを通知します。シークレットの名前にスタック名を含める必要があります。次の例で、スタック名は「hub」です。

```
docker secret create hub_WEBSERVER_CUSTOM_CERT_FILE <certificate file>
```

```
docker secret create hub_WEBSERVER_CUSTOM_KEY_FILE <key file>
```

2. docker-compose.local-overrides.ymlファイルでwebserverサービスにシークレットを追加します。

```
webserver:
  secrets:
    - WEBSERVER_CUSTOM_CERT_FILE
    - WEBSERVER_CUSTOM_KEY_FILE
```

3. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルの末尾にあるsecretsセクションからコメント文字(#)を削除します。

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

4. docker-compose.local-overrides.ymlファイルのwebserverサービスのhealthcheckプロパティは、シークレットからの新しい証明書をポイントしている必要があります。

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

5. 次のコマンドを実行して、スタックを再導入します。

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

トラブルシューティング

次のエラーが発生した場合は、次の手順に従ってください。

Error response from daemon: rpc error: code = AlreadyExists desc = secret hub_WEBSERVER_CUSTOM_CERT_FILE already exists.

1. Black Duckを停止します。

```
docker stack rm hub
docker ps : to wait until all containers are down
```

2. 以前のシークレットを削除します。

```
docker secret rm hub_WEBSERVER_CUSTOM_CERT_FILE
docker secret rm hub_WEBSERVER_CUSTOM_KEY_FILE
```


3. 新しい有効なシークレットで、シークレットを再び作成します。

```
docker secret create hub_WEBSERVER_CUSTOM_CERT_FILE <certificate file>
docker secret create hub_WEBSERVER_CUSTOM_KEY_FILE <key file>
```

4. Black Duckを再デプロイします。

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

5. nginxを含むすべてのコンテナが正常になるまで待ちます。

 注：すでに証明書を更新し、オーバーライドファイルに変更を加えている場合は、Black Duckの新しいバージョンには新しいファイルが含まれるため、シークレット部分のコメントを必ず解除してください。

証明書認証でのカスタム認証局の使用

証明書認証に独自の認証局を使用できます。

カスタム認証局を使用するには、次の手順を実行します。

1. カスタム認証局証明書ファイルであるAUTH_CUSTOM_CAという名前のdocker secretを、docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルのwebserverおよびauthenticationサービスに追加します。

```
webserver:
  secrets:
    - AUTH_CUSTOM_CA
authentication:
  secrets:
    - AUTH_CUSTOM_CA
```

2. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルの末尾に次のテキストを追加します。

```
secrets:
  AUTH_CUSTOM_CA:
    file: {file path on host machine}
```

3. webserverコンテナとauthenticationサービスを開始します。
4. Black Duckサービスが起動したら、信頼できる認証局で署名された証明書キー ペアを含むJson Webトークン (JWT)を返すAPIリクエストを行います。以下に例を示します。

```
curl https://localhost:443/jwt/token --cert user.crt --key user.key
```

 注：認証に使用される証明書のユーザー名は、Black DuckシステムにCommon Name (CN、コモン ネーム) として存在する必要があります。

ナレッジベースからのコンポーネント移行データの取得

移行APIを使用し、Black Duck KnowledgeBaseから新規または変更されたコンポーネントIDを取得します。これらのAPIは、ナレッジベース内の移行されたコンポーネントの生データと新しいIDを返します。

APIを使用して、古いコンポーネントID (例: コンポーネントID: bf368a1d-ef4f-422c-baca-a138737595e7)を送信し、ナレッジベースから新しいコンポーネントIDを取得できます。

移行追跡APIは、特定のコンポーネントまたはコンポーネントバージョンの移行情報を取得したり、特定の日付に発生した移行の詳細を取得したりできます。

移行の記録の有効化

移行の記録を有効にするには、次で以下のプロパティを設定します: blackduck-config.env file.

RECORD_MIGRATIONS = true (デフォルトはfalse)

これにより、移行が検出されたときにレコードを書き込むことができます。

移行データの保持

レコードが返される日数を設定するには、blackduck-config.envファイルで次のプロパティを設定します。

MIGRATED_OBJECT_RETENTION_DAYS = <number_of_days> (デフォルトは30日)

APIエンドポイント

APIドキュメントに移動して、次のAPIの使用を開始します。

特定の日付以降に発生した移行の場合：

```
GET /api/component-migrations
```

特定のコンポーネントまたはコンポーネントバージョンの移行の場合：

```
GET /api/component-migrations/{componentOrVersionId}
```

詳細については、『Black Duck APIドキュメント』(https://<blackduck_server>/api-doc/public.html#_component_component_version_migration_endpoints)を参照してください。

無視されたコンポーネントをレポートに含める

デフォルトでは、無視されたコンポーネントとそれらの無視されたコンポーネントに関連付けられた脆弱性は、脆弱性ステータスレポート、脆弱性更新レポート、脆弱性修正レポート、およびプロジェクトバージョンレポートから除外されます。無視されたコンポーネントを含めるには、docker-swarmディレクトリにあるblackduck-config.envファイルでBLACKDUCK_REPORT_IGNORED_COMPONENTS環境変数の値を「true」に設定します。


BLACKDUCK_REPORT_IGNORED_COMPONENTSの値を「false」にリセットすると、無視されたコンポーネントが除外されます。

セキュリティ保護されたLDAPの設定

セキュリティで保護されたLDAPサーバーをBlack Duckに接続するときに証明書の問題が発生する場合は、セキュリティで保護されたLDAPサーバーへの信頼関係接続をBlack Duckサーバーが設定していないことが考えられます。これは通常、自己署名証明書を使用している場合に発生します。

セキュリティで保護されたLDAPサーバーの信頼関係接続を設定するには、次の方法でローカルBlack Duck LDAPトラストストアにサーバー証明書をインポートします。

1. LDAP情報を取得します。
2. Black DuckのUIを使用してサーバー証明書をインポートします。

 **注：** ホストされるすべてのお客様は、SAMLまたはLDAPを介したシングル サインオン (SSO) の既成サポートを活用して、Black Duckアプリケーションへのアクセスのセキュリティを確保する必要があります。これらのセキュリティ機能を有効にして設定する方法については、インストールガイドを参照してください。さらに、2要素認証を提供しているSAML SSOプロバイダを使用しているお客様は、そのテクノロジーを有効にして活用し、Black Duckアプリケーションへのアクセスのセキュリティをさらに高めることをお勧めします。

LDAP情報の取得

LDAP管理者に連絡し、次の情報を収集します。

LDAPサーバーの詳細

Black Duck SCAがディレクトリサーバーに接続するために使用する情報です。

- ・ (必須) インスタンスがリスンするディレクトリサーバーのホスト名またはIPアドレス(プロトコルスキームとポートを含む)です。
例: `ldaps://<server_name>.<domain_name>.com:339`
- ・ (オプション) 組織で匿名認証を使用せず、LDAPアクセスに認証情報が必要な場合は、ディレクトリサーバーの読み取り権限を持っているユーザーのパスワードと、LDAP名または絶対LDAP識別名(DN)のいずれかです。
絶対LDAP DNの例: `uid=ldapmanager,ou=employees,dc=company,dc=com`

LDAP名の例: `jdoe`

4. 管理タスク・セキュリティ保護されたLDAPの設定

- ・ (オプション)LDAPアクセスに認証情報が必須である場合は、使用する認証タイプ (simpleまたはdigest-MD5) です。

LDAPユーザー属性

ディレクトリサーバー内のユーザーを見つけるためにBlack Duckで使用する情報です。

- ・ (必須)ユーザーを見つけるために使用できる絶対ベースDNです。
例: dc=example,dc=com
- ・ (必須)特定の一意ユーザーとのマッチングに使用される属性です。この属性の値により、その名前のユーザーのユーザープロフィールアイコンがパーソナライズされます。
例: uid=[0]

テスト用ユーザー名とパスワード

- ・ (必須)ディレクトリサーバーへの接続をテストするためのユーザーの認証情報

サーバー証明書のインポート

サーバー証明書をインポートするには、次の手順を実行します。

1. システム管理者としてBlack Duckにログインします。

2.  をクリックします。

3. [統合] → [外部認証] を選択します。

4. [ライトウェイト ディレクトリ アクセス プロトコル(LDAP)] をクリックします。

5. [LDAP構成を有効にする]チェックボックスをオンにして、上で説明したように[LDAPサーバーの詳細]セクションに情報を入力します。[サーバーURL]フィールドで、セキュリティで保護されたLDAPサーバー (プロトコルスキームはldaps://) が構成されていることを確認します。

6. 上記の説明に従って、[LDAPユーザー属性]セクションに情報を入力します。

必要に応じて、[Black Duckで自動的にユーザー アカウントを作成する]チェックボックスをオフにして、LDAPで認証されたユーザーの自動作成をオフにします。このチェックボックスはデフォルトでオンになっているため、Black Duckに存在しないユーザーがLDAPを使用してBlack Duckにログインすると、ユーザーが自動で作成されます。これは、新規インストールとアップグレードに適用されます。

7. [テスト接続、ユーザー認証およびフィールドマッピング]セクションにユーザーの認証情報を入力し、[接続のテスト]をクリックします。
8. 証明書に問題がない場合は、自動的にインポートされ、「接続テストに成功しました」というメッセージが表示されます。

Test Connection, User Authentication and Field Mapping

Tests ability to connect and authenticate test-user. Note: test-user credentials are not saved.

✓ Connection Test Succeeded

Test Username *

Denis Bors

Test Password *

Reset

Test Connection

9. 証明書に問題がある場合、ダイアログボックスに証明書に関する詳細が表示されます。次のいずれかを実行します。
 - ・ [キャンセル]をクリックして、証明書の問題を解決します。
解決したら、接続を再度テストし、証明書の問題が解決して証明書がインポートされたことを確認します。成功した場合は、「接続テストに成功しました」というメッセージが表示されます。
 - ・ [保存]をクリックしてこの証明書をインポートします。
[接続のテスト]をクリックして、証明書がインポートされたことを確認します。成功した場合は、「接続テストに成功しました」というメッセージが表示されます。

LDAP信頼ストアのパスワード

カスタムのBlack Duck Webアプリケーション信頼ストアを追加する場合は、これらの方法を使用してLDAP信頼ストアのパスワードを指定します。

Docker Swarmを使用する場合は、次の方法を使用してLDAP信頼ストアのパスワードを指定します。

- ・ `docker secret`コマンドでLDAP_TRUST_STORE_PASSWORD_FILEを使用してDocker Swarmにパスワードを通知します。シークレットの名前にスタック名を含める必要があります。この例では、「HUB」がスタック名です。
`docker secret create HUB_LDAP_TRUST_STORE_PASSWORD_FILE <file containing password>`
`docker-swarm`ディレクトリにある`docker-compose.local-overrides.yml`ファイルで、パスワードシークレットをwebappサービスに追加します。


```
secrets:
  - LDAP_TRUST_STORE_PASSWORD_FILE
```

`docker-compose.local-overrides.yml`ファイルの末尾にあるsecrets セクションに、次のようなテキストを追加します。

```
secrets:
  LDAP_TRUST_STORE_PASSWORD_FILE:
    external: true
    name: "HUB_LDAP_TRUST_STORE_PASSWORD_FILE"
```
- ・ `docker-swarm`ディレクトリにある`docker-compose.local-overrides.yml`ファイルにwebappサービスのボリュームセクションを追加して、LDAP_TRUST_STORE_PASSWORD_FILEというファイルが含まれているディレクトリを/run/secretsにマウントします。

4. 管理タスク・ログファイルとヒートマップデータのダウンロード

```
webapp:
  volumes: ['/directory/where/file/is:/run/secrets']
```

 注：信頼ストアが完全に置き換えられ、別のパスワードで保護されている場合にのみ、LDAP_trust_store_password_fileが含まれているディレクトリをマウントする必要があります。


ログファイルとヒートマップデータのダウンロード

問題のトラブルシューティングやカスタマサポートへのログファイルの提供が必要になる場合があります。システム管理者の役割のあるユーザーは、現在のログファイルまたはシステムのヒートマップデータが格納されている.zipファイルをダウンロードできます。

ログファイルまたはヒートマップファイルの準備には数分かかる場合があることに注意してください。ログ取得とヒートマップデータの設定の詳細については、『インストールガイド』を参照してください。

ログファイルへのアクセス


Black DuckのUIからログファイルをダウンロードするには、次の手順を実行します。

1. システム管理者の役割でBlack Duckにログインします。
2.  → [ジョブ] の順にクリックします。
3. [システム情報]タブを選択します。
4. [直近 2 日分のログ (.zip)] または [直近 14 日分のログ (.zip)] のいずれかをクリックします。

ヒートマップデータへのアクセス

ヒートマップを圧縮CSVとしてダウンロードして端末スキャンの傾向を確認および分析し、スプレッドシートプログラムのピボットとしてヒートマップを作成することができます。

システムのヒートマップデータをダウンロードするには：

1. システム管理者の役割でBlack Duckにログインします。
2.  → [ジョブ] の順にクリックします。
3. [システム情報]タブを選択します。
4. [ヒートマップ データ] セクションで [ヒートマップのダウンロード(.zip)] をクリックします。

ログファイルの表示

ログの取得

コンテナからログを取得するには、次の手順を実行します。

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

ここで、「logs/」は、ログのコピー先のローカルディレクトリです。

コンテナのログファイルの表示

docker-compose logsコマンドを使用して、すべてのログを表示します。

```
docker-compose logs
```

Dockerコマンドについて詳しくは、DockerドキュメントWebサイトを参照してください。 <https://docs.docker.com/>

ログのページ

デフォルトでは、ログファイルは14日後に自動的にページされます。この値を変更するには、次の手順を実行します。


1. コンテナを停止します。
2. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルを編集します。
 - a. logstashサービスを追加します。
 - b. 新しい値でDAYS_TO_KEEP_LOGS環境変数を追加します。次の例では、10日後にログファイルがページされます。

```
logstash:
  environment: {DAYS_TO_KEEP_LOGS: 10}
```

3. コンテナを再起動します。

デフォルトのメモリ制限の変更

Black Duckにかかる負荷によっては、一部のコンテナでデフォルトのメモリ制限よりも高いメモリ制限が必要になることがあります。

 **注：** デフォルトのメモリ制限は、絶対に引き下げないでください。引き下げると、Black Duckが正しく機能しなくなります。

次のコンテナのデフォルトのメモリ制限を変更できます。

- ・ webapp
- ・ jobrunner
- ・ scan
- ・ binaryscanner
- ・ bomengine

webappコンテナのデフォルトのメモリ制限の変更

webappコンテナには、次の3つのメモリ設定があります。

- ・ HUB_MAX_MEMORY環境変数は、最大Javaヒープサイズを制御します。
- ・ limits memoryとreservations memory設定は、Dockerがwebappコンテナの全体的なメモリをスケジュールおよび制限するのに使用する制限を制御します。
 - ・ limits memory設定は、コンテナが使用できるメモリ容量です。
 - ・ Dockerはreservations memory設定を使用して、コンテナをマシンに導入（スケジュール）できるかどうかを判断します。この値を使用して、Dockerは、すべてのコンテナが同じメモリを求めて競合するのではなく、マシンに導入されたすべてのコンテナに十分なメモリがあることを確認します。

これらの各設定の値は、最大Javaヒープサイズよりも大きくする必要があります。Black Duck Softwareでは、Javaヒープサイズを更新する場合は、limits memoryとreservations memoryの値をそれぞれ最大Javaヒープサイズよりも少なくとも1 GB大きく設定することを推奨しています。

docker-compose.local-overrides.ymlファイルのwebappセクションを使用し、必要に応じてコメント文字(#)を削除して、新しい値を入力します。

4. 管理タスク・デフォルトのメモリ制限の変更

次の例では、Web Appコンテナの最大Javaヒープサイズが8 GBに変更され、limit memoryとreservations memory設定の値がそれぞれ9 GBに変更されます。

元の値:

```
#webapp:
#environment: {HUB_MAX_MEMORY: REPLACE_WITH_NEW_MAX_MEMORYm}
#deploy:
  #limits: {MEMORY: REPLACE_WITH_NEW_MEM_LIMITm}
  #reservations: {MEMORY: REPLACE_WITH_NEW_VALUEm}
```

更新された値:


```
webapp:
environment: {HUB_MAX_MEMORY: 8192m}
deploy:
  limits: {MEMORY: 9216m}
  reservations: {MEMORY: 9216m}
```

jobrunnerコンテナのデフォルトのメモリ制限の変更

jobrunnerコンテナには、次の3つのメモリ設定があります。

- HUB_MAX_MEMORY環境変数は、最大Javaヒープサイズを制御します。
- limits memoryとreservations memory設定は、Dockerがjobrunnerコンテナの全体的なメモリをスケジュールおよび制限するのに使用する制限を制御します。
 - limits memory設定は、コンテナが使用できるメモリ容量です。
 - Dockerはreservations memory設定を使用して、コンテナをマシンに導入(スケジュール)できるかどうかを判断します。この値を使用して、Dockerは、すべてのコンテナが同じメモリを求めて競合するのではなく、マシンに導入されたすべてのコンテナに十分なメモリがあることを確認します。

これらの各設定の値は、最大Javaヒープサイズよりも大きくする必要があります。Black Duck Softwareでは、Javaヒープサイズを更新する場合は、limits memoryとreservations memoryの値をそれぞれ最大Javaヒープサイズよりも少なくとも1 GB大きく設定することを推奨しています。

 注: これらの設定は、スケーリングされたJob Runnerコンテナを含む、すべてのJob Runnerコンテナに適用されます。

docker-compose.local-overrides.ymlファイルのjobrunnerセクションを使用し、必要に応じてコメント文字(#)を削除して、新しい値を入力します。

次の例では、jobrunnerコンテナの最大Javaヒープサイズが8 GBに変更され、limit memoryとreservations memory設定の値がそれぞれ9 GBに変更されます。

元の値:

```
#jobrunner:
#environment: {HUB_MAX_MEMORY: REPLACE_WITH_NEW_MAX_MEMORYm}
#deploy:
  #limits: {MEMORY: REPLACE_WITH_NEW_MEM_LIMITm}
  #reservations: {MEMORY: REPLACE_WITH_NEW_VALUEm}
```

更新された値:

```
jobrunner:
environment: {HUB_MAX_MEMORY: 8192m}
deploy:
  limits: {MEMORY: 9216m}
  reservations: {MEMORY: 9216m}
```

デフォルトのメモリ制限の変更

HUB_MAX_MEMORY値を変更するときにコンテナにメモリを追加した場合、次の値が変更されることがあります。


- `org.quartz.scheduler.periodic.maxthreads: 4`
- `org.quartz.scheduler.periodic.prefetch: 2`
- `org.quartz.scheduler.ondemand.maxthreads: 8`
- `org.quartz.scheduler.ondemand.prefetch: 4`


ガイドンスの例

```
SMALL
HUB_MAX_MEMORY: 4096m
org.quartz.scheduler.periodic.maxthreads: 4
org.quartz.scheduler.periodic.prefetch: 2
org.quartz.scheduler.ondemand.maxthreads: 8
org.quartz.scheduler.ondemand.prefetch: 4

Medium
HUB_MAX_MEMORY: 6144m
org.quartz.scheduler.periodic.maxthreads: 4
org.quartz.scheduler.periodic.prefetch: 2
org.quartz.scheduler.ondemand.maxthreads: 12
org.quartz.scheduler.ondemand.prefetch: 8

Large/X-Large
HUB_MAX_MEMORY: 12288m
org.quartz.scheduler.periodic.maxthreads: 8
org.quartz.scheduler.periodic.prefetch: 4
org.quartz.scheduler.ondemand.maxthreads: 24
org.quartz.scheduler.ondemand.prefetch: 16
```

 注: LargeとX-Largeの違いは、jobrunnerインスタンスの数だけです。メモリとスレッドの数は同じです。


 注: 両方のプールの最大スレッド数は32を超えてはいけません

scanコンテナのデフォルトのメモリ制限の変更

scanコンテナには、次の3つのメモリ設定があります。

- `HUB_MAX_MEMORY`環境変数は、最大Javaヒープサイズを制御します。
- `limits memory`と`reservations memory`設定は、DockerがScanコンテナの全体的なメモリをスケジュールおよび制限するのに使用する制限を制御します。
 - `limits memory`設定は、コンテナが使用できるメモリ容量です。
 - Dockerは`reservations memory`設定を使用して、コンテナをマシンに導入(スケジュール)できるかどうかを判断します。この値を使用して、Dockerは、すべてのコンテナが同じメモリを求めて競合するのではなく、マシンに導入されたすべてのコンテナに十分なメモリがあることを確認します。

これらの各設定の値は、最大Javaヒープサイズよりも大きくする必要があります。Black Duck Softwareでは、Javaヒープサイズを更新する場合は、`limits memory`と`reservations memory`の値をそれぞれ最大Javaヒープサイズよりも少なくとも1 GB大きく設定することを推奨しています。

 注: これらの設定は、スケーリングされたScanコンテナを含む、すべてのScanコンテナに適用されます。

`docker-compose.local-overrides.yml`ファイルの`scan`セクションを使用し、必要に応じてコメント文字(#)を削除して、新しい値を入力します。

次の例では、scanコンテナの最大Javaヒープサイズが4 GBに増やされ、`limits memory`と`reservations memory`設定の値がそれぞれ5 GBに増やされます。

元の値:

```
#scan:
#environment: {HUB_MAX_MEMORY: REPLACE_WITH_NEW_MAX_MEMORYm}
#deploy:
```

4. 管理タスク・デフォルトのメモリ制限の変更


```
#limits: {MEMORY: REPLACE_WITH_NEW_MEM_LIMITm}
#reservations: {MEMORY: REPLACE_WITH_NEW_VALUEm}
```

更新された値:

```
scan:
environment: {HUB_MAX_MEMORY: 4096m}
deploy:
  limits: {MEMORY: 5210m}
  reservations: {MEMORY: 5210m}
```

binaryscannerコンテナのデフォルトのメモリ制限の変更

binaryscannerコンテナの唯一のデフォルトのメモリサイズは、コンテナの実際のメモリ制限です。

 注: これらの設定は、スケーリングされたbinaryscannerコンテナを含む、すべてのbinaryscannerコンテナに適用されます。

docker-compose.local-overrides.ymlファイルにbinaryscannerセクションを追加します。

次の例では、コンテナのメモリ制限が4 GBに変更されます。


更新された値:

```
binaryscanner:
  limits: {MEMORY: 4096M}
```

bomengineコンテナのデフォルトのメモリ制限の変更

bomengineコンテナには、次の3つのメモリ設定があります。

- HUB_MAX_MEMORY環境変数は、最大Javaヒープサイズを制御します。
- limits memoryとreservations memory設定は、Dockerがbomengineコンテナの全体的なメモリをスケジュールおよび制限するのに使用する制限を制御します。
 - limits memory設定は、コンテナが使用できるメモリ容量です。
 - Dockerはreservations memory設定を使用して、コンテナをマシンに導入(スケジュール)できるかどうかを判断します。この値を使用して、Dockerは、すべてのコンテナが同じメモリを求めて競合するのではなく、マシンに導入されたすべてのコンテナに十分なメモリがあることを確認します。

 注: これらの設定は、スケーリングされたbomengineコンテナを含む、すべてのbomengineコンテナに適用されます。

docker-compose.local-overrides.ymlファイルのbomengineセクションを使用し、必要に応じてコメント文字(#)を削除して、新しい値を入力します。

次の例では、bomengineコンテナの最大Javaヒープサイズが8 GBに変更され、limits memoryとreservations memory設定の値がそれぞれ9 GBに変更されます。

元の値:

```
#bomengine:
#environment: {HUB_MAX_MEMORY: REPLACE_WITH_NEW_MAX_MEMORYm}
#deploy:
  #limits: {MEMORY: REPLACE_WITH_NEW_MEM_LIMITm}
  #reservations: {MEMORY: REPLACE_WITH_NEW_VALUEm}
```

更新された値:

```
bomengine:
environment: {HUB_MAX_MEMORY: 8192m}
deploy:
  limits: {MEMORY: 9216m}
  reservations: {MEMORY: 9216m}
```


- ❗ 重要：Black Duck bomengineコンテナのHUB_MAX_MEMORYとlimits memoryには、jobrunnerコンテナと同じ値を割り当てることをお勧めします。

logstashのホスト名の変更

logstashのホスト名とサービス名を変更しても、内部PostgreSQLコンテナにはプッシュされません。たとえば、logstashを別の目的で使用していて、PostgreSQLログをbdlogstashに書き込むために、logstashホスト名をbdlogstashに変更したい場合は、次の手順を実行します。

1. blackduck-config.envで、logstashをbdlogstashに変更します：HUB_LOGSTASH_HOST=bdlogstash。
2. docker-compose.ymlを編集して、logstashをbdlogstashに変更します。

bdlogstash:

```
image: blackducksoftware/blackduck-logstash:1.0.9
volumes: ['log-volume:/var/lib/logstash/data']
env_file: [blackduck-config.env]
```

3. docker-compose.ymlのpostgresコンテナ定義にenv_file: [blackduck-config.env]を追加して、ホスト名の変更が読み取られるようにします。

env_file: [blackduck-config.env]

マップされていないコードの場所のクリーンアップ

マップされていないコードの場所は、プロジェクトバージョンにマップされていないコードの場所です。Black Duckの内部では、コードの場所は1つ以上のスキャンによって表されます。


- ・ 場合によっては、コードスキャンが、コードの場所をプロジェクトバージョンにマッピングせずにコードの場所を作成することがあり、この結果、マップされていないコードの場所になります。
- ・ プロジェクトバージョンを削除すると、コードの場所/スキャンデータが孤立する可能性があります。

マップされていないコードの場所のデータをパージしたくない場合は、blackduck-config.envファイルでこの機能を無効にすることができます。これはデフォルトでtrueに設定されており、30日ごとにパージするように設定されています。

定期的にスキャンし、頻繁にデータを破棄したい場合は、保持期間を14日間などの短い日数に設定することをお勧めします。

マップされていないコードの場所のクリーンアップをスケジュールするには、blackduck-config.envファイルで次のプロパティを設定します。

- ・ BLACKDUCK_HUB_UNMAPPED_CODE_LOCATION_CLEANUP=true
デフォルト設定はtrueです。
- ・ BLACKDUCK_HUB_UNMAPPED_CODE_LOCATION_RETENTION_DAYS=<number of days between 1-365, for example, 14>
デフォルト設定は30日です。


 注：マップされていないコードの場所のクリーンアップを構成してシステムを再起動すると、スキャンパージが開始され、保持条件(構成された保持日数よりも古い)を満たすマップされていないコードの場所が削除されます。デフォルトでは、スキャンパージジョブは15分ごとに実行されます。

cron文字列を使用したスキャンページジョブのスケジュール

docker swarm実装のblackduck-config.envファイルに変数を設定することで、スキャンページを構成できます。

次のいずれかの方法を使用して、blackduck-config.envで変数を設定することにより、スキャンページジョブをスケジュールできます。

- ・ cronジョブを使用する場合: blackduck.scan.processor.scanpurge.cronstring
次のcron形式を使用します。second minute hour dayOfMonth month daysOfWeek
- ・ 固定遅延を使用する場合: blackduck.scan.processor.scanpurge.fixeddelay (scanpurgejobを実行する頻度をミリ秒単位で設定します。デフォルトは15分です)

 注: blackduck.scan.processor.scanpurge.cronstringを指定した場合、代わりにcron文字列が使用されるため、blackduck.scan.processor.scanpurge.fixeddelay設定は無視されます。

スタックしている構成表イベントのクリア

構成表イベントクリーンアップジョブは、処理エラーのためにスタックしている可能性がある構成表イベントをクリアします。デフォルトでは、ジョブは毎日午前0時に実行され、過去24時間より前にスタックしたイベントを削除します。cronスケジュールは、システムの使用率が低い時間帯に応じて変更できます。また、過去何時間のイベントを保持するかを設定を1~48時間の間で変更できます。

- ・ 過去何時間の構成表イベントを保持するかを設定します。これは、処理エラーまたはトポロジ変更によってスタックしている可能性のある構成表イベントをページするのに役立ちます。デフォルトは24時間です。有効な範囲は1~48時間です。たとえば、過去12時間より前にスタックしたすべてのイベントを削除する場合は、次のように指定します。

```
BLACKDUCK_BOM_EVENT_CLEANUP_BEFORE_HOURS=12
```

- ・ Cron式を介して、構成表イベントをクリアするジョブの実行スケジュールを設定します。システムの使用率が低い時間帯に実行することをお勧めします。デフォルトでは、午前0時に実行され、値は0 0 * * * ?です。たとえば、構成表イベントクリーンアップジョブを毎日午前2時に実行する必要がある場合は、次のように指定します。

```
BLACKDUCK_BOM_EVENT_CLEANUP_JOB_CRON_TRIGGER="0 2 * * * ?"
```

VersionBomEventCleanupJobはデフォルトで有効になっており、このジョブを無効にすることはできません。

上書きファイルの使用

Black Duckで使用するデフォルト設定の一部を上書きすることができます。.yamlファイルを直接編集する代わりに、docker-swarmディレクトリにあるdocker-compose.local-overrides.yamlを使用します。

このファイルを使用してデフォルト設定を変更することで、アップグレード時に変更内容が保持され、Black Duckをアップグレードするたびに.yamlファイルを変更する必要がなくなります。

docker-composeコマンドで、docker-compose.local-overrides.yamlファイルは最後に使用される.yamlファイルである必要があります。たとえば、次のコマンドは、外部データベースを使用してBlack Duckを起動します。

```
docker stack deploy -c docker-compose.externaldb.yaml -c docker-compose.local-overrides.yaml hub
```

分析の構成: Black Duck


Black Duckでは、blackduck-config.envファイルで解析をオフにすることで、Black Duck Detectの自動実行をグローバルに無効にできます。

1. blackduck-config.envファイルで、ANALYTICS=falseのように構成します。

2. Black Duckを再起動します。

外部PostgreSQLインスタンスの構成

Black Duck は、外部PostgreSQLインスタンスの使用をサポートしています。Black Duck 2024.10.x の場合、PostgreSQL バージョン 15.x、16.x がサポートされます。新規インストールの場合は、Black Duck では 16.x シリーズの最新バージョンを推奨しています。

 注： Azure PostgreSQLを使用してインストールする場合、データベース管理者は、2023.4.0以降をインストールまたはこれ以降にアップグレードする前に、hstore PostgreSQL拡張機能のインストールを有効にする必要があります。さらに、init スクリプトが Azure データベースにアクセスできるよう、azure.extensions 値に pgcrypto と hstore 両方が含まれていることを確認してください。

外部 PostgreSQL データベースを設定するには、次の手順に従います。

1. 外部PostgreSQLクラスタをUTF8エンコードで初期化します。これを行う方法は、外部PostgreSQLプロバイダによって異なる場合があります。たとえば、PostgreSQL initdbツールを使用する場合は、次のコマンドを実行します。

```
initdb --encoding=UTF8 -D /path/to/data
```

2. データベースのユーザー名とパスワードを作成および設定します。PostgreSQLデータベースには、管理者（デフォルトでは、blackduckがユーザー名）、ユーザー（デフォルトでは、blackduck_userがユーザー名）、Black Duckレポート データベースのユーザー（デフォルトでは、blackduck_reporterがユーザー名）の3種類のユーザーが存在します。次の操作を実行できます。

- ・ デフォルトのユーザー名でアカウントを作成する。
- ・ カスタムユーザー名でアカウントを作成する。

3. PostgreSQLインスタンスを構成します。

デフォルトのユーザー名を使用したアカウントの作成と構成：

デフォルトのblackduck、blackduck_user、およびblackduck_reporterのユーザー名を使用する場合は、次の手順を実行します。

これらの手順を完了したら、「[PostgreSQLインスタンスの構成](#)」に進みます。

1. 管理者権限を持つblackduckという名前のデータベースユーザーを作成します。

Amazon RDSの場合は、データベースインスタンスの作成時に「マスターユーザー」をblackduckに設定します。

その他の特定の値は必要ありません。


2. ディレクトリにあるexternal-postgres-init.pgsqlファイルで以下を docker-swarm

置き換えます。

POSTGRES_USERを次と置き換えます： blackduck

HUB_POSTGRES_USERを次と置き換えます： blackduck_user

BLACKDUCK_USER_PASSWORDを次に使用するパスワードに置き換えます： blackduck_user

 重要： この手順は必須です。


3. docker-swarmディレクトリにあるexternal-postgres-init.pgsqlスクリプトを実行して、ユーザー、データベース、およびその他の必要なアイテムを作成します。以下に例を示します。

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

4. 任意のPostgreSQL管理ツールを使用して、blackduck、blackduck_user、およびblackduck_reporterデータベースユーザーのパスワードを構成します。

これらのユーザーは、前の手順でexternal-postgres-init.pgsqlスクリプトによって作成されています。

5. 「PostgreSQLインスタンスの構成」に進みます。

 注：Black Duck はpgcrypto拡張機能を使用し、それが利用可能であると想定します。CloudSQL、RDS、およびAzureは、デフォルトでpgcrypto拡張機能を提供しているため、さらなる構成は必要ありません。Community PostgreSQLのユーザーは、postgresql-contribパッケージがまだインストールされていない場合は、それをインストールする必要があります。パッケージ名はディストリビューションによって異なります。

カスタムユーザー名とパスワードを使用したアカウントの作成と構成：

カスタムデータベースのユーザー名を作成する場合は、次の手順を実行します。

次の手順で、各ユーザー名は次のとおりです。

- ・ DBAdminNameは、新しいカスタム管理者のユーザー名。
- ・ DBUserNameは、新しいカスタムデータベースユーザーのユーザー名。
- ・ DBReporterNameは、新しいカスタムデータベースレポーターのユーザー名。

数字のみで構成されるユーザー名は、外部PostgreSQLインスタンスのPostgreSQLユーザー名に使用できます。

更新されたexternal-postgres-init.pgsqlスクリプトでは、数値のPostgreSQLユーザー名がスクリプトで正常に実行されるように、ユーザー名を二重引用符で囲みます。external-postgres-init.pgsqlスクリプトで、HUB_POSTGRES_USERおよびPOSTGRES_USERのデフォルトの名前値を検索して、数値のユーザー名に置き換えます。

これらの手順を完了したら、次のセクション「PostgreSQLインスタンスの構成」に進みます。

1. 管理者権限を持つDBAdminNameという名前のデータベースユーザーを作成します。

Amazon RDSの場合は、データベースインスタンスの作成時に「マスターユーザー」をDBAdminNameに設定します。

その他の特定の値は必要ありません。

2. docker-swarmディレクトリにあるexternal-postgres-init.pgsqlスクリプトを編集し、DBAdminName、DBUserName、およびDBReporterNameに使用するアカウント名を指定します。
3. docker-swarmディレクトリにある編集済みのexternal-postgres-init.pgsqlスクリプトを実行して、ユーザー、データベース、およびその他の必要なアイテムを作成します。以下に例を示します。

```
psql -U DBAdminName -h <hostname> -p <port> -f external-postgres-init.pgsql postgres
```
4. 任意のPostgreSQL管理ツールを使用して、DBAdminName、DBUserName、およびDBReporterNameデータベースユーザーのパスワードを構成します。

これらのユーザーは、前の手順でexternal-postgres-init.pgsqlスクリプトによって作成されています。

5. hub-postgres.env環境ファイルを編集します。このファイルには、HUB_POSTGRES_USERおよびHUB_POSTGRES_ADMINのデフォルトのユーザー名がリストされています。これらのデフォルト値を、データベースユーザーおよび管理者のカスタムユーザー名に置き換えます。
6. 次のセクション「PostgreSQLインスタンスの構成」に進みます。

PostgreSQLインスタンスの構成：

ユーザーを作成してパスワードを構成したら、次の手順を実行します。

1. docker-swarmディレクトリにあるhub-postgres.env環境ファイルを編集して、データベース接続パラメータを指定します。次のことを行えます。
 - ・ データベース接続でSSLを有効にする。

認証に、証明書またはユーザー名とパスワード、あるいはその両方を使用することを選択できます。

- ・ データベース接続でSSLを無効にする。

SSLが無効の場合は、ユーザー名とパスワードによる認証を使用する必要があります。

パラメータ	説明
HUB_POSTGRES_ENABLE_SSL	データベース接続でのSSLの使用を定義します。 <ul style="list-style-type: none"> ・ データベース接続でSSLを使用しないように設定するには、値を「false」に設定します。これがデフォルト値です。 ・ データベース接続でSSLを使用するように設定するには、値を「true」に設定します。
HUB_POSTGRES_ENABLE_SSL_CERT	認証に証明書が必要かどうかを定義します。 <ul style="list-style-type: none"> ・ クライアント証明書認証を無効にするには、値を「false」に設定します。これがデフォルト値です。 ・ データベース接続でSSLを使用するときにクライアント証明書認証を要求する場合は、値を「true」に設定します。
HUB_POSTGRES_HOST	PostgreSQLインスタンスを含むサーバーのホスト名。
HUB_POSTGRES_PORT	PostgreSQLインスタンスの接続先のデータベースポート。

2. ユーザー名とパスワードによる認証を使用している場合は、PostgreSQL管理者とユーザーのパスワードをBlack Duckに提供します。

- a. データベースユーザーのパスワードが含まれたHUB_POSTGRES_USER_PASSWORD_FILEという名前のファイルを作成します。デフォルトのユーザー名を使用している場合、これはblackduck_userのユーザー名で、前の例ではDBUserNameです。
- b. データベース管理者ユーザーのパスワードが含まれたHUB_POSTGRES_ADMIN_PASSWORD_FILEという名前のファイルを作成します。デフォルトのユーザー名を使用している場合、これはblackduckのユーザー名で、前の例ではDBAdminNameです。
- c. 両方のファイルが含まれているディレクトリを/run/secretsにマウントします。docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルを使用します。各サービス(webapp、jobrunner、authentication、bomengine、およびscan)で、次の手順を実行します。
 1. 必要に応じて、サービス名の前にあるコメント文字(#)を削除します。
 2. サービスにマウントするボリュームを追加します。

次の例では、webappサービスにボリュームが追加されます。

```
webapp:
  volumes: ['directory/of/password/files:/run/secrets']
```

またこのテキストを、authentication、jobrunner、bomengine、scanの各サービスへ追加する必要があります。

手順2a～cの代わりにdocker secretコマンドを使用して、シークレットHUB_POSTGRES_USER_PASSWORD_FILEとシークレットHUB_POSTGRES_ADMIN_PASSWORD_FILEを作成できます。

- a. docker secretコマンドを使用して、Docker Swarmにシークレットを通知します。シークレットの名前にスタック名を含める必要があります。次の例で、スタック名は「hub」です。

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

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


4. 管理タスク・外部PostgreSQLインスタンスの構成

- b. `docker-compose.local-overrides.yml`ファイルで、`webapp`、`jobrunner`、`authentication`、`bomengine`、および `scan` サービスにパスワードシークレットを追加します。`webapp` サービスの例を次に示します。

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

必要に応じて、コメント文字(#)を削除します。

コメント文字を削除し、必要に応じてスタック名を`docker-compose.local-overrides.yml`ファイル末尾のテキストに変更します。

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

3. 証明書認証を使用している場合は、`docker-swarm` ディレクトリにある `docker-compose.local-overrides.yml` ファイルを編集し、`webapp`、`jobrunner`、`authentication`、`scan` サービスで、すべての証明書ファイル (`HUB_POSTGRES_CA` (サーバー CA ファイル)、`HUB_POSTGRES_CRT` (クライアント証明書ファイル)、`HUB_POSTGRES_KEY` (クライアント キー ファイル)) が含まれているディレクトリを `/run/secrets` にマウントします。手順2cの例を参照してください。
4. 証明書および/またはユーザー名/パスワード認証でSSLを使用できるようにするには、`HUB_POSTGRES_ENABLE_SSL_CERT`を「true」に設定し、手順2と3を完了します。
5. Black Duckを[インストール](#)または[アップグレード](#)します。

既存の外部データベースのPostgreSQLユーザー名の変更

デフォルトでは、PostgreSQLデータベースユーザーのユーザー名は`blackduck_user`で、PostgreSQL管理者のユーザー名は`blackduck`です。

外部PostgreSQLデータベースを使用している場合は、これらのユーザー名を変更できます。

これらの手順は、外部データベースが現在`blackduck`および`blackduck_user`ユーザー名を使用している既存のBlack Duckインスタンス用の手順です。外部データベースの新しい構成のユーザー名を変更するには、前のセクションの手順に従います。

! 重要： 管理者権限を持っていないBlack Duckデータベースユーザー(これはGCPやRDSなどのホストされたプロバイダで一般的)の場合は、`bds_hub`データベースに接続して次を実行します `GRANT blackduck_user TO blackduck;`

既存のPostgreSQLアカウント名を変更するには、次の手順を実行します。

1. [Black Duckを停止](#)します。
2. `bds_hub`データベースで、ユーザーの名前を変更し、パスワードをリセットします。

```
alter user blackduck_user rename to NewName1 ;
alter user blackduck rename to NewName2 ;
alter user NewName1 password 'NewName1Password' ;
alter user NewName2 password 'NewName2Password' ;
```
3. `docker-swarm`ディレクトリにある`hub-postgres.env`ファイルで、`HUB_POSTGRES_USER`および `HUB_POSTGRES_ADMIN`の値を編集します。`HUB_POSTGRES_USER`の値は、`blackduck_user`の新しいユーザー名です。`HUB_POSTGRES_ADMIN`の値は、`blackduck`の新しいユーザー名です。以下に例を示します。

```
HUB_POSTGRES_USER=NewName1
HUB_POSTGRES_ADMIN=NewName2
```

4. Black Duckを再起動します。

 注：2020.4.0リリースでは、BDIOデータベースが削除されました。

プロキシ設定の構成

blackduck-config.envファイルを編集して、プロキシ設定を構成します。外部インターネットアクセスにプロキシが必要な場合は、これらの設定を構成する必要があります。

Black Duck Softwareがホストするサービスへのアクセスに必要なコンテナは次のとおりです。

- ・ Authentication
- ・ Registration
- ・ Job runner
- ・ Web app
- ・ スキャン
- ・ Bomengine

プロキシ環境変数は次のとおりです。

- ・ HUB_PROXY_HOST。プロキシサーバーホストの名前。
- ・ HUB_PROXY_PORT。プロキシサーバーホストがリスンするポート。
- ・ HUB_PROXY_SCHEME。プロキシサーバーへの接続に使用するプロトコル。
- ・ HUB_PROXY_USER。プロキシサーバーにアクセスためのユーザー名。

NTLMプロキシの環境変数は次のとおりです。

- ・ HUB_PROXY_WORKSTATION。認証要求発信元のワークステーション。基本的には、このマシンのコンピュータ名。
- ・ HUB_PROXY_DOMAIN。内部で認証するドメイン。

ReversingLabs プロキシのサポート

ReversingLabsのプロキシ サポートは制限されており、高度な構成はサポートされていません。環境変数HUB_PROXY_*を使用し、ホストとポート、またはホスト、ポート、ユーザー名、パスワードのみでプロキシを指定できます。

プロキシパスワード

認証がプロキシ経由で使用される場合、次のサービスではプロキシパスワードが必要です。

- ・ Authentication
- ・ Bomengine
- ・ Web App
- ・ Registration
- ・ Job Runner
- ・ スキャン

プロキシパスワードを指定する方法として、次の3つの方法があります。

4. 管理タスク・プロキシ設定の構成

- ・ テキストファイルHUB_PROXY_PASSWORD_FILEが含まれているディレクトリを、/run/secretsへマウントします。これは最も安全なオプションです。
- ・ プロキシパスワードを含む環境変数HUB_PROXY_PASSWORDを指定します。
- ・ 以下の手順に従い、docker secretコマンドを使用してシークレットHUB_PROXY_PASSWORD_FILEを作成します。
 1. docker secretコマンドを使用して、Docker Swarmにシークレットを通知します。シークレットの名前をスタック名に含める必要があります。次の例で、スタック名は「hub」です。

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

2. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルで、各サービス (authentication、webapp、registration、jobrunner、Match engine、Bom engine、およびscan) に対してシークレットへのアクセスを提供します。scanサービスの例を次に示します。

```
scan:
  secrets:
    - HUB_PROXY_PASSWORD_FILE
```

必要に応じて、コメント文字(#)を削除します。


3. docker-compose.local-overrides.ymlファイルの末尾にあるsecretsセクションに、次を追加します。

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

必要に応じて、コメント文字(#)を削除します。

環境変数が個別のマウント済みのファイルまたはシークレットで指定されていない場合は、blackduck-config.envファイルを使用して環境変数を指定できます：

1. HUB_PROXY_PASSWORDの前にあるシャープ記号(#)を削除し、コメントアウトを解除します。
2. プロキシパスワードを入力します。
3. ファイルを保存します。

 注： Docker Swarm導入でdocker secret HUB_PROXY_PASSWORD_FILEを使用してプロキシパスワードを提供したときにKB呼び出しが失敗した場合は、blackduck-config.envファイルでパスワードを提供して問題を解決します。

プロキシ証明書のインポート

プロキシ証明書をインポートして、プロキシを操作できます。

1. プロキシ証明書ファイルを使用し、docker secret <stack name>_HUB_PROXY_CERT_FILEを作成します。次に例を示します。

```
docker secret create <stack name>_HUB_PROXY_CERT_FILE <certificate file>
```

2. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルで、次のサービスにシークレットへのアクセスを提供します：authentication、webapp、registration、jobrunner、およびscan。scanサービスの例を次に示します。

```
scan:
  secrets:
    - HUB_PROXY_CERT_FILE
```

認証済みプロキシの使用


JDK 8u111([統合済みJDK 8リリース ノート\(oracle.com\)](#))に加えられた変更により、基本認証が必要なプロキシを使用するお客様は、Black Duck登録サービスとの通信に問題が生じる場合があります。この問題を解決するには、次の変更をblackduck-config.env(Docker Swarm)またはConfigMap(Kubernetes/OpenShift)へ加える必要があります:

```
REGISTRATION_SERVICE_OPTS: "-Djdk.http.auth.tunneling.disabledSchemes='"
```

レポートデータベースのパスワードの設定

このセクションでは、レポートデータベースのパスワードを構成する手順について説明します。

docker-swarm/binディレクトリにあるhub_reportdb_changepassword.shスクリプトを使用して、レポートデータベースのパスワードを設定または変更します。

 注: このスクリプトは、Black Duckによって自動的にインストールされるデータベースコンテナを使用している場合に、レポートデータベースのパスワードを設定または変更します。外部のPostgreSQLデータベースを使用している場合は、任意のPostgreSQL管理ツールを使用してパスワードを設定します。

スクリプトを実行してパスワードを設定または変更する場合は、次の点に注意してください。

- dockerグループのユーザーまたはルートユーザーであるか、sudoアクセス権を持っている必要があります。
- PostgreSQLデータベースコンテナを実行しているDockerホストで操作を実行する必要があります。

次の例では、レポートデータベースのパスワードは「blackduck」に設定されます。

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

job runner、scan、bomengine、およびbinaryscannerコンテナのスケーリング

job runner、scan、bomengine、およびbinaryscannerコンテナをスケーリングできます。

次のコマンドを実行するには、dockerグループのユーザーまたはルートユーザーであるか、sudoアクセス権を持っている必要があります。


bomengineコンテナのスケーリング

この例では、2つ目のbomengineコンテナが追加されます。

```
docker service scale hub_bomengine=2
```

bomengineコンテナの現在の数に満たない数を指定することで、bomengineコンテナを削除できます。次の例では、bomengineコンテナが減らされて1つになります。

```
docker service scale hub_bomengine=1
```

 注: Black Duck bomengineコンテナをjobrunnerコンテナと同じレベルにスケーリングすることをお勧めします。

job runnerコンテナのスケーリング

この例では、2つ目のJob Runnerコンテナが追加されます。

```
docker service scale hub_jobrunner=2
```

job runnerコンテナの現在の数未満の数を指定することで、job runnerコンテナを削除できます。次の例では、job runnerコンテナが減らされて1つになります。

4. 管理タスク・シングルサインオンのSAMLの設定

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

scanコンテナのスケールリング

この例では、2つ目のScanコンテナが追加されます。

```
docker service scale hub_scan=2
```

scanコンテナの現在の数未満の数を指定することで、scanコンテナを削除できます。次の例では、scanコンテナが減らされて1つになります。


```
docker service scale hub_scan=1
```

binaryscannerコンテナのスケールリング

binaryscannerコンテナはBlack Duck Binary Analysisで使用されます。

この例では、2つ目のbinaryscannerコンテナが追加されます。

```
docker service scale bdba-worker=2
```

 注：Binaryscannerコンテナを1個追加するごとに、1 CPU、2 GB RAM、100 GBの空きディスク容量の追加が必要です。

binaryscannerコンテナの現在の数よりも小さい数を指定することで、binaryscannerコンテナを削除できます。次の例では、binaryscannerコンテナが減らされて1つになります。

```
docker service scale bdba-worker=1
```

シングルサインオンのSAMLの設定

Security Assertion Markup Language (SAML) は、XMLベースのオープンな標準データ形式で、パーティ間で認証および認証データを交換します。たとえば、アイデンティティプロバイダとサービスプロバイダ間での交換です。Black DuckのSAML実装ではシングルサインオン(SSO)機能が提供され、SAMLが有効になっているとき、Black DuckユーザーはBlack Duckに自動的にサインインできます。SAMLの有効化はBlack Duckユーザー全員に適用され、個別のユーザーに選択的に適用することはできません。

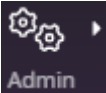
ホストされるすべてのお客様は、SAMLまたはLDAPを介したシングルサインオン(SSO)の既成サポートを活用して、Black Duckアプリケーションへのアクセスのセキュリティを確保する必要があります。これらのセキュリティ機能を有効にして設定する方法については、インストールガイドを参照してください。さらに、2要素認証を提供しているSAML SSOプロバイダを使用しているお客様は、そのテクノロジーを有効にして活用し、Black Duckアプリケーションへのアクセスのセキュリティをさらに高めることをお勧めします。

次の点に注意してください。

- SAMLとLDAPの両方を同時に設定することはできません。
- SAML機能の有効/無効を切り替えるには、システム管理者の役割を持つユーザーである必要があります。
- Black Duck は属性ステートメントで情報が提供されている場合、外部ユーザーの情報(氏名、名、姓、電子メール)を同期して取得できます。姓と名の値は大文字と小文字が区別されるため注意してください。
Black Duck は、Black Duckでグループ同期を有効にした場合、外部ユーザーのグループ情報を同期することもできます。
- SAMLを有効にしてログインすると、Black Duckのログインページではなく、Identity Providerのログインページにリダイレクトされます。
- SSOユーザーがBlack Duckからログアウトしたときに、Black Duckから正常にログアウトしたことを通知するログアウトページが表示されるようになりました。このログアウトページには、Black Duckに再ログインするためのリンクが含まれ、ユーザーがBlack Duckに正常に再ログインするために資格情報を提供する必要がありません。

- ・ SSOシステムに問題があり、SSO設定を無効にする必要がある場合は、Black Duck servername/sso/loginを入力し、Black Duckにログインします。

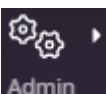
SAMLを使用するシングルサインオンの有効化または無効化

1.  をクリックします。
2. [統合] → [外部認証] を選択します。
3. [Security Assertion Markup Language (SAML)] をクリックし、次の手順に従います。
 - a. [SAML構成を有効にする]チェックボックスをオンにします。
 - b. [サービスプロバイダのエンティティID]フィールド: ご使用の環境のBlack Duckサーバーの情報を、https://host形式で入力します。ここで、hostはBlack Duckサーバーです。
 - c. 次のIdentity Providerメタデータのいずれかを選択します。
 - ・ [URL]を選択して、Identity ProviderのURLを入力します。
 - ・ [XMLファイル]を選択して、ファイルをドロップするか、表示されている領域をクリックして、XMLファイルを選択できるダイアログボックスを開きます。
 - d. [サービスプロバイダのエンティティID]フィールド。ご使用の環境のBlack Duckサーバーの情報を、https://host形式で入力します。ここで、hostはBlack Duckサーバーです。
 - e. 外部 Black Duck URL フィールド。Black DuckサーバーのパブリックURLです。
例: https://blackduck-docker01.dc1.lan
 - f. 必要に応じ、次のいずれかを選択します。
 - ・ 署名付き認証要求を送信: このオプションが有効になっている場合、管理されているユーザーは、送信前に認証要求に署名することを、管理者ユーザーから求められています。
 - ・ グループ同期化を有効にする: このオプションが有効な場合、ログイン時にIdentity Provider (IDP) のグループがBlack Duckで作成され、ユーザーはこれらのグループに割り当てられます。「Groups」という属性名を使用して属性ステートメントでグループを送信するように、IDPを設定する必要があります。
 - ・ ローカルログアウトのサポートの有効化: このオプションが有効な場合、Black Duckからログアウトすると、IDPのログインページが表示されます。ローカルログアウトのサポートが有効な場合、SAML要求はForceAuthn="true"で送信されます。IDPをチェックして、この機能がサポートされていることを確認します。
 - ・ Black Duckで自動的にユーザー アカウントを作成する: このオプションがオンになっている環境で、ユーザーがIdPを使用してログインし、Black Duck内にそのユーザーの登録がなかった場合、Black Duckのデータベースにローカル ユーザーが作成されます。

4. [保存]をクリックします。

[保存]をクリックすると、[Black DuckメタデータURL]フィールドが表示されます。リンクをコピーするか、SAML XML構成情報を直接ダウンロードできます。

SAMLを使用するシングルサインオンを無効にするには

1.  をクリックします。
2. [統合] → [外部認証] を選択します。
3. [Security Assertion Markup Language (SAML)] をクリックします。

4. 管理タスク・ソースファイルのアップロード

4. [SAML 設定を有効にする] チェックボックスをオフにします。
5. [保存] をクリックします。

補足情報

- ・ Assertion Consumer Service (ACS) : `https://<host>/saml/SSO`
- ・ 推奨されるサービスプロバイダのエンティティID: `https://<host>`。hostはBlack Duckサーバーの場所です。

ソースファイルのアップロード

構成表レビュー担当者は、マッチを確認し、偽陰性を調査することで、スキャンの結果を簡単に確認できる必要があります。スニペットマッチをレビューする場合、ソースファイルとマッチの並列比較を確認することはマッチの評価とレビューに役立ちます。

Black Duck では、ソースファイルをアップロードすることができます。これにより、構成表レビュー担当者はBlack Duck UI内からファイルの内容を確認することができます。

スキャン中にディープ ライセンス データの検出または著作権テキストの検索を有効にした場合、ソース ファイルをアップロードすると、構成表レビュー担当者は、検出されたライセンスまたは著作権テキストをBlack Duck UI内で確認できます。ファイルがアップロードされると、Black Duckは埋め込みライセンスと著作権テキストのリストを提供し、ファイル内のテキストが強調表示で表示されます。

構成表レビュー担当者がBlack Duck UI内からファイル コンテンツを表示するには、次の手順を実行します。

1. 管理者は、ソースファイルをアップロードできるようにする必要があります。
 - a. 管理者は、環境変数を使用してこの機能を有効にします。
 - b. 管理者はオプションで、**シークレットの暗号化を設定**します。ホストされている顧客のソース コードのアップロードは常に暗号化されます。
2. スキャン クライアントは、SSL/TLSで保護されたエンドポイントと適切な認証トークンを使用し、ソース ファイルの内容をBlack Duckインスタンスに送信します。

その後、ファイルは**暗号化されます**。アップロードされたファイルは、ファイル名ではなく、関連するスキャンIDとファイル署名を使用して保存されます。

Black Duck UIで、ソースファイルは、HTTPSを介してネットワーク経由で送信されます。

次の点に注意してください。

- ・ ファイルをアップロードするのに必要な十分なディスク容量があることを確認してください。
- ・ 一度にアップロードできるソースの最大合計サイズは4 GB (4,000 MB) です。この値は構成可能です。
- ・ アップロードされたファイルは180日後に削除されます。この値は構成可能です。
- ・ アップロードサービスが最大ディスク設定の95%に達すると、ファイルは削除されます。
サービスは、ディスク容量が最大ディスク設定の90%になるまで、古いファイルから順に削除します。


ファイルのアップロードを有効にする

ファイルのアップロードを有効にするには、docker-swarmディレクトリにあるblackduck-config.envファイルの環境変数ENABLE_SOURCE_UPLOADSをtrueに設定します。

```
ENABLE_SOURCE_UPLOADS=true
```

ソースコードの暗号化を有効にする

ストレージ サービスで管理されている他の機密データとともにアップロードされたソースコードの暗号化を有効にするには、SYNOPTSYS_CRYPT0_ENABLEDを設定する必要があります。シークレット暗号化の詳細については、「[Black Duckでのシークレット暗号化の構成](#)」を参照してください。

 注：ホストされている顧客のソースコードのアップロードは常に暗号化されます。

起動または停止 Black Duck

これらのコマンドを使用して、Black Duckを起動またはシャットダウンします。

起動 Black Duck

上書きファイルを使用してデフォルトの構成設定を変更していない場合は、これらのコマンドを使用します。

- PostgreSQLデータベースコンテナを使用してBlack Duckを起動するには、次のコマンドを実行します。

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

- PostgreSQLデータベースコンテナを使用してBlack DuckをBlack Duck Binary Analysisとともに起動するには、次のコマンドを実行します。

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

- 外部データベースを使用してBlack Duckを起動するには、次のコマンドを実行します。

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

- 外部データベースを使用してBlack DuckをBlack Duck Binary Analysisとともに起動するには、次のコマンドを実行します。

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


- Swarmサービスに対してファイル システムが読み取り専用の状態でBlack Duckを実行している場合は、前の手順にdocker-compose.readonly.ymlファイルを追加します。

たとえば、PostgreSQLデータベースコンテナを使用してBlack Duckをインストールするには、次のコマンドを入力します。

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

上書きファイル使用時のBlack Duckの起動

上書きファイルを使用してデフォルトの構成設定を変更した場合は、これらのコマンドを使用します。

 注：docker-compose.local-overrides.ymlファイルは、docker-composeコマンドで最後に使用される.ymlファイルである必要があります。

- PostgreSQLデータベースコンテナを使用してBlack Duckを起動するには、次のコマンドを実行します。

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

- PostgreSQLデータベースコンテナを使用してBlack DuckをBlack Duck Binary Analysisとともに起動するには、次のコマンドを実行します。

```
docker swarm init
```


4. 管理タスク・ユーザーセッションタイムアウトの構成

```
docker stack deploy -c docker-compose.yml -c docker-compose.bdba.yml -c docker-  
compose.local-overrides.yml hub
```

- ・ 外部データベースを使用してBlack Duckを起動するには、次のコマンドを実行します。

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

- ・ 外部データベースを使用してBlack DuckをBlack Duck Binary Analysisとともに起動するには、次のコマンドを実行します。

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

- ・ Swarmサービスに対してファイル システムが読み取り専用の状態でBlack Duckを実行している場合は、前の手順にdocker-compose.readonly.ymlファイルを追加します。

たとえば、PostgreSQLデータベースコンテナを使用してBlack Duckをインストールするには、次のコマンドを入力します。

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

シャットダウン Black Duck

- ・ 次のコマンドを実行してBlack Duckをシャットダウンします。


```
docker stack rm hub
```

ユーザーセッションタイムアウトの構成

Black Duckサーバーからユーザーを自動的にログアウトし、企業のセキュリティポリシーに合わせるには、ユーザーセッションタイムアウト値を構成します。

1. 現在のタイムアウト値を表示するには、次のGETリクエストを行います。


```
GET https://<Black-Duck-server>/api/system-oauth-client
```

 注：GETメソッドを使用するには、OAuthクライアントの読み取り権限を持っている必要があります。

2. 現在のタイムアウト値を変更するには、PUTリクエスト本文で次のPUTリクエストを行います。

```
PUT https://<Black-Duck-server>/api/system-oauth-client
```

```
{  
  "accessTokenValiditySeconds": <time value in seconds>  
}
```

 注：このタスクでPUTメソッドを使用するには、OAuthクライアントを更新する権限を持っている必要があります。システム管理者の役割には、必要な権限が含まれています。

PUTリクエスト本文に入力する値は、新しいタイムアウト値です。

30分(1,800秒)から24時間(86,400秒)までのタイムアウト値を使用できます。

次のメディアタイプを使用できます。

```
application/vnd.blackducksoftware.user-4+json  
application/json
```

Postmanの例を次に示します。

PUT https://localhost/api/system-oauth-client	
Params	Authorization ● Headers (10) Body ● Pre-request Script Tests
▼ Headers (2)	
KEY	VALUE
<input checked="" type="checkbox"/> Accept	application/vnd.blackducksoftware.user-4+json
<input checked="" type="checkbox"/> Content-Type	application/vnd.blackducksoftware.user-4+json

次の例では、タイムアウト値が7,200秒から8,000秒に変更されます。

The screenshot shows a REST client interface with a PUT request to `https://<hub-server>/api/system-oauth-client...`. The request body is a JSON object with the following structure:

```

1 {
2   "accessTokenValiditySeconds": 8000
3 }

```

The response body is also a JSON object, showing the updated client information:

```

1 {
2   "clientId": "00000000-0000-4000-0000-000000000001",
3   "scopes": [
4     "READ",
5     "CLIENT_MANAGEMENT",
6     "WRITE"
7   ],
8   "grantTypes": [
9     "IMPLICIT"
10  ],
11  "registeredRedirecturis": [],
12  "accessTokenValiditySeconds": 7200,
13  "refreshTokenValiditySeconds": 14400,
14  "_meta": {
15    "allow": [
16      "DELETE",
17      "GET",
18      "PUT"
19    ],
20    "href": "https://.../api/oauthclients/00000000-0000-4000-0000-000000000001",
21    "links": []
22  }
23 }

```

An arrow points from the `"accessTokenValiditySeconds": 8000` in the request body to the `"accessTokenValiditySeconds": 7200` in the response body, with a note: "Change 7200 to 8000 seconds by using PUT request with the new value."

カスタマサポートへのお客様のBlack Duckシステム情報の提供

カスタマサポートは、システム統計と環境またはネットワーク情報など、Black Duckのインストールに関する情報の提供を求める場合があります。この情報をすばやく簡単に取得できるようにするために、Black Duckでは、この情報を収集するためのsystem_check.shスクリプトを提供しています。スクリプトは、この情報を作業ディレクトリにあるsystem_check.txtファイルに出力します。その後、このファイルをカスタマサポートに送信できます。

system_check.shスクリプトはdocker-swarm/binディレクトリにあります。

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

このスクリプトを実行するには、dockerグループのユーザーまたはルート ユーザーであるか、sudoアクセス権を持っている必要があります。

デフォルトのsysadminユーザーについて

Black Duck SCAをインストールすると、デフォルトのシステム管理者(sysadmin)アカウントが構成されます。デフォルトのsysadminユーザーには、すべての役割と権限が関連付けられています。

i ヒント: ベストプラクティスとして、最初のログインにはデフォルトのsysadminアカウントを使用し、すぐにデフォルトのパスワード(blackduck)を変更して、サーバーのセキュリティを確保する必要があります。パスワードを変更するには、Black DuckのUIの右上隅にあるユーザー名/ユーザープロフィールアイコンから[マイプロフィール]を選択します。

sysadminユーザーに関連付けられているデフォルトの電子メールアドレスを編集するには、Black DuckのUIの[ユーザーの管理]ページに移動し、sysadminユーザー名を選択し、電子メールアドレスを変更して保存します。変更を確認するには、ログアウトしてから再度ログインする必要があります。

[ユーザーの管理]ページにアクセスするには、使用するユーザーアカウントにユーザー管理者の役割が必要です。この役割は、デフォルトでsysadminアカウントに割り当てられています。電子メールアドレスの主な目的は、ユーザーアカウントの連絡先参照情報として使用することです。

Black Duckレポート遅延の構成

Black Duck 2024.10.0で、レポートデータベースジョブプロセスは480分ごとに実行されます。これは構成可能です。

別のレポート遅延を設定するには、次の手順を実行します。

1. docker-swarmディレクトリのblackduck-config.envファイルを編集し、を構成します。
BLACKDUCK_REPORTING_DELAY_MINUTES=<value in minutes>

例: BLACKDUCK_REPORTING_DELAY_MINUTES=360
2. コンテナを再起動します。

コンテナのタイムゾーンの構成

デフォルトでは、Black DuckコンテナのタイムゾーンはUTCです。監視目的で、ログに表示されるタイムスタンプにローカルタイムゾーンが反映されるようにこの値を変更することができます。

別のタイムゾーンを設定するには、次の手順を実行します。

1. docker-swarmディレクトリにあるblackduck-config.envファイルのTZ環境変数の値を新しいタイムゾーンに設定します。[ここ](#)に示すように、Wikipediaに示されている値を使用します。

たとえば、コロラド州デンバーで使用されているタイムゾーンに変更するには、次のように入力します。
TZ=America/Denver
2. コンテナを再起動します。

デフォルトの使用法の変更

使用法は、このバージョンがリリースされるときにコンポーネントをプロジェクトにどのように含めるかについて示します。

使用法の値は次のとおりです。

- ・ 静的にリンク。静的にリンクされ、プロジェクトとともに配布される、密接に統合されているコンポーネント。
- ・ 動的にリンク。DLLやjarファイルなど、動的にリンクされ、中程度に統合されているコンポーネント。

- ・ ソースコード。javaまたは.cppファイルなどのソースコード。
- ・ 開発ツール/除外。コンポーネントは、リリースされたプロジェクトに含まれません。たとえば、構築、開発、またはテストのため社内的に使用されるコンポーネントです。例として、ユニットテスト、IDEファイル、コンパイラがあります。
- ・ 分離した製品。統合の弱いコンポーネントを対象としています。作業内容はコンポーネントから取得されていません。分離した製品と見なされるようにするには、アプリケーションに独自の実行可能ファイルが含まれていて、コンポーネントとアプリケーションがリンクされていない必要があります。例としては、配布メディアに無償のAcrobat PDFビューアを含めている場合などがあります。
- ・ 標準の実装。標準に従って実装している場合を対象としています。たとえば、プロジェクトとともに出荷されるJava Spec Requestなどがあります。
- ・ 単純集合。プロジェクトで使用されていない、または依存していないコンポーネントを対象としています。ただし、これらは同じメディア上にある場合があります。たとえば、関連付けられていない製品のサンプル版を配布に含めている場合があります。
- ・ 前提条件。必要ですが配布で提供されないコンポーネントを対象としています。
- ・ 未指定。このコンポーネントの使用法はまだ明確にされていません。

Black Duckバージョン2020.10.0で、Black DuckはUNSPECIFIED使用法値を導入しました。既存のデフォルトの代わりに、これをデフォルト オプションとして使用することで、コンポーネントに正しい使用法値またはデフォルトの使用法値のどちらかが割り当てられているかどうかの混乱を解消できます。たとえば、デフォルトの使用法値としてDYNAMICALLY_LINKEDを使用した場合、DYNAMICALLY_LINKEDの正しい使用法値を持つコンポーネントと、デフォルトで使用方法値が割り当てられたコンポーネントを区別できない場合があります。使用法のデフォルトとしてUNSPECIFIEDを使用することで、使用方法値としてUNSPECIFIEDが割り当てられていることが表示された場合に、有効な使用方法値を割り当てるためのアクションを実行する必要があることがわかります。

デフォルトの使用法は、マッチタイプによって決まります。スニペットの使用法はソースコードですが、その他のすべてのマッチタイプの使用法は動的にリンクです。

Black Duck には次の変数があり、これらを使用して類似するマッチタイプのデフォルトの使用法を変更できます。

- ・ BLACKDUCK_HUB_FILE_USAGE_DEFAULT。この変数の使用法を定義すると、次のマッチタイプのデフォルト値が設定されます。
 - ・ バイナリ
 - ・ 完全ディレクトリ
 - ・ 完全ファイル
 - ・ 追加/削除されたファイル
 - ・ 変更したファイル
 - ・ 部分
- ・ BLACKDUCK_HUB_DEPENDENCY_USAGE_DEFAULT。この変数の使用法を定義すると、次のマッチタイプのデフォルト値が設定されます。
 - ・ ファイルの依存関係
 - ・ 直接的な依存関係
 - ・ 推移的な依存関係
- ・ BLACKDUCK_HUB_SOURCE_USAGE_DEFAULT。この変数の使用法を定義すると、次のマッチタイプのデフォルト値が設定されます。
 - ・ スニペット

4. 管理タスク・Black DuckコンテナのユーザーIDのカスタマイズ

- ・ BLACKDUCK_HUB_MANUAL_USAGE_DEFAULT。この変数の使用法を定義すると、次のマッチタイプのデフォルト値が設定されます。
 - ・ 手動で追加
 - ・ 手動で判定
- ・ BLACKDUCK_HUB_SHOW_UNMATCHED: マッチしなかったコンポーネント数を表示するかどうかを決定します。デフォルトはfalseです(表示されません)。

アップロードされたjsonld/bdioファイルbdio2のマッチタイプ

これらのマッチのデフォルトの使用法は、動的にリンクです。

- ・ 直接的な依存関係バイナリ
- ・ 推移的な依存関係バイナリ

これらは、既存のバイナリマッチと同じフルファイルマッチです。


1つのファイルに複数の推移的な依存関係バイナリマッチを含めることができますが、直接的な依存関係バイナリは1つのみです。

別の使用法値を構成するには、次の手順を実行します。

1. docker-swarmディレクトリにあるblackduck-config.envファイルで、コメントアイコン(#)を削除し、値を入力して、新しい使用法値にします。ファイルに示されている使用法値のいずれかを使用します: SOURCE_CODE、STATICALLY_LINKED、DYNAMICALLY_LINKED、SEPARATE_WORK、IMPLEMENTATION_OF_STANDARD、

たとえば、ファイルのデフォルトの使用法を未指定に変更するには、次のようにします。

```
BLACKDUCK_HUB_FILE_USAGE_DEFAULT=UNSPECIFIED
```

 注: 正しくない使用法テキストを入力した場合は、元のデフォルト値が適用されます。jobrunnerコンテナのログファイルに警告メッセージが表示されます。

変更された使用法値は、新しいスキャンまたは再スキャンに適用されます。

Black DuckコンテナのユーザーIDのカスタマイズ

コンテナを実行するユーザーID (UID) の変更が必要になる場合があります。

各コンテナの現在のUIDは次のとおりです。

- ・ Authentication (blackduck-authentication) : 100
- ・ Binaryscanner (bdba-worker) : 0
- ・ CA (blackduck-cfssl) : 100
- ・ DB (blackduck-postgres) : 1001
- ・ Documentation (blackduck-documentation) : 8080
- ・ Job Runner (blackduck-jobrunner) : 100
- ・ Logstash (blackduck-logstash) : 100
- ・ RabbitMQ (rabbitmq) : 100
- ・ Registration (blackduck-registration) : 8080
- ・ Scan (blackduck-scan) : 8080
- ・ Web App (blackduck-webapp) : 8080

- ・ Webserver(blackduck-nginx) : 100
- ・ Redis(blackduck-redis) : 任意のユーザー/グループとして実行
- ・ Bomengine(blackduck-bomengine) : 100
- ・ Matchengine(blackduck-matchengine) : 100

UIDを変更するには、docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlにコンテナの新しい値を追加します。コンテナのセクションに

user:UID_NewValue:root 行を追加します。

次の例では、webappコンテナのUIDを1001に変更します。

webapp:

user: 1001:root

次の点に注意してください。

- ・ postgresコンテナとbinaryscannerコンテナのUIDは変更できません。
 - ・ postgresコンテナのUIDは1001でなければなりません。
 - ・ binaryscannerコンテナのUIDは0(ルート)でなければなりません。
- ・ 一部のコンテナのUID値は同じですが(たとえば、Documentation、Registration、Scan、Web AppコンテナのUIDは8080)、1つのコンテナのUID値を変更しても、同じUID値のコンテナのUID値は変更されません。たとえば、Web Appコンテナの値を8080から1001に変更しても、Documentation、Scan、またはRegistrationコンテナの値は変更されません。これらのコンテナのUID値は8080のままになります。
- ・ コンテナは、どのユーザーとしてコンテナが実行されるかを問わず、ユーザーはルートグループに属するものとして指定される必要があると想定します。

UIDをカスタマイズするには、次の手順を実行します。

1. Black Duckを[停止します](#)。
2. 上記のように値を編集します。
3. Black Duckを[起動します](#)。

Webサーバー設定の構成

hub-webserver.envファイルを次のように編集します。

- ・ ホスト名を構成します。
- ・ ホストポートを構成します。
- ・ IPv6を無効にします。

ホスト名の構成

hub-webserver.envファイルを編集して、証明書ホスト名が一致するようにホスト名を設定します。環境変数には、デフォルト値としてサービス名が設定されています。

証明書が構成されていない場合、Webサーバーの起動時にHTTPS証明書が生成されます。ホスト名が一致するように、PUBLIC_HUB_WEBSEVER_HOST環境変数の値を指定して、リッスンするホスト名をWebサーバーに通知する必要があります。そうしないと、証明書にはホスト名として使用するサービス名のみが含まれるようになります。この値は、ユーザーがBlack Duckにアクセスするためにブラウザに入力する公開ホスト名に変更する必要があります。以下に例を示します。

4. 管理タスク・UTF8文字エンコードを使用した構成表レポートの作成

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

ホストポートの構成

ホストポートには、異なる値を設定できます。デフォルトは443です。

ホストポートを設定するには、次の手順を実行します。

1. docker-swarmディレクトリにあるdocker-compose.local-overrides.ymlファイルに、新しいホストポート値を追加します。

webserverセクションを使用して、次の形式でポート情報を追加します。ports: ['NewValue:8443'] たとえば、ポートを8443に変更するには、次のように指定します。

```
webserver:
  ports: [ '8443:8443' ]
```

2. hub-webserver.envファイルのPUBLIC_HUB_WEBSERVER_PORT値を編集して、新しいポート値を追加します。以下に例を示します。

```
PUBLIC_HUB_WEBSERVER_PORT=8443
```

3. docker-compose.ymlファイルの次の部分をコメントアウトして、ホストのポート443へのアクセスを無効にします。そうしないと、ユーザーは引き続きポート443を使用してホストにアクセスできます。

```
webserver:
  #ports: [ '443:8443' ]
```

IPv6の無効化

デフォルトでは、NGiNXはIPv4とIPv6をリッスンします。ホストマシンでIPv6が無効になっている場合は、IPV4_ONLY環境変数の値を1に変更します。

UTF8文字エンコードを使用した構成表レポートの作成

アルファベット以外の文字を使用している場合に構成表レポートでUTF8文字エンコードのサポートを有効にするには、blackduck-config.envファイルに以下を追加します。

```
USE_CSV_BOM=true
```

スキャン監視

スキャン監視エンドポイントの情報が収集されるデフォルトの時間は1時間です。この時間は、次のプロパティを使用して別の値に設定できます。

HUB_SCAN_MONITOR_ROLLUP_WINDOW_SECONDS

上記で設定した最後の時間(デフォルトの期間は1時間)に、システムが分析するスキャンが一定数以上ある場合のみAPIは情報を提供できます。スキャンがしきい値よりも小さい場合、レベル1リクエストはデフォルトでOK応答を返します。スキャン数は、次のプロパティを使用して別の値に設定できます。

HUB_SCAN_MONITOR_MINIMUM_SCAN_COUNT

レベル1リクエストは、失敗率のしきい値に基づいて「OK」または「NOT OK」の応答を返します(上限を超えると、「NOT OK」応答が返されます)。上限(デフォルトは30%)と下限(デフォルトは10%)はパーセンテージで設定され、次のプロパティで設定できます。

HUB_SCAN_MONITOR_ERROR_RATE_LOWER_THRESHOLD_PERCENT

HUB_SCAN_MONITOR_ERROR_RATE_HIGHER_THRESHOLD_PERCENT

スキャン監視APIリクエストの詳細については、Black Duckで入手可能な『REST API開発者ガイド』を参照してください。

HTMLレポートのダウンロードサイズの設定

任意のサイズのHTMLレポートをダウンロードできるように環境を設定できます。デフォルトのサイズは3000 KBです。この制限を超えるレポートは、エラーメッセージ「503サービスは使用できません」が返されます。

この制限を変更するには、blackduck-config.envファイルのHUB_MAX_HTML_REPORT_SIZE_KBプロパティの値を変更します。

KBライセンス更新ジョブおよびセキュリティ更新ジョブの構成

KBライセンス更新ジョブおよびセキュリティ更新ジョブを無効にするには、blackduck-config.envファイルで次のプロパティを追加します。

```
KB_UPDATE_JOB_ENABLED=FALSE
```

KBライセンス更新ジョブの頻度を変更するには、blackduck-config.envファイルで次のプロパティを追加します。


```
KB_LICENSE_UPDATER_PERIOD_MINUTES=<分単位の時間>
```

シークレット暗号化の構成：Black Duck

Black Duck は、システム内の重要なデータの保存時の暗号化をサポートします。この暗号化は、オーケストレーション環境 (Docker Swarm または Kubernetes) によって Black Duck インストールにプロビジョニングされたシークレットに基づいています。次に、組織のセキュリティポリシーに基づいてこのシークレットを作成および管理し、バックアップシークレットを作成し、シークレットをローテーションするプロセスの概要を示します。

暗号化される重要なデータは、次のとおりです。

- ・ SCM統合OAuthトークン
- ・ SCM統合プロバイダOAuthアプリケーションクライアントシークレット
- ・ LDAP認証情報
- ・ SAMLプライベート署名証明書
- ・ RSAキー

 注：シークレットの暗号化は、いったん有効にすると、無効にすることはできません。

暗号化シークレットの概要

暗号化シークレットは、システム内のリソースをアンロックする目的で内部暗号化キーを生成するために使用されるランダムなシーケンスです。Black Duckのシークレットの暗号化は、3つの対称キー、つまりルートキー、バックアップキーおよび以前のキーによって制御されます。これらの3つのキーは、KubernetesおよびDocker SwarmシークレットとしてBlack Duckに渡されたシードによって生成されます。3つのシークレットは、次のように名前が付けられます。

- ・ crypto-root-seed
- ・ crypto-backup-seed
- ・ crypto-prev-seed

通常の状態では、3つのシードはすべて、アクティブな使用中にはなりません。ローテーションアクションが進行中ではない限り、アクティブな唯一のシードはルートシードになります。

ルートシードのセキュリティ保護

ルートシードを保護することは重要です。システムデータのコピーとともにルートシードを所有するユーザーは、システムの保護されたコンテンツをアンロックし、読み取る可能性があります。一部のDocker SwarmシステムまたはKubernetesシステムは、デフォルトでは、保存時のシークレットを暗号化しません。これらのオーケストレーションシステムを内部で暗号化するように構成して、後でシステムに作成されるシークレットが安全に保たれるようにすることを強くお勧めします。

ルートシードは、災害復旧計画の一部としてバックアップからシステム状態を再作成するのに必要です。ルートシードファイルのコピーは、オーケストレーションシステムとは別の秘密の場所に保存して、シードとバックアップの組み合わせでシステムを再作成できるようにする必要があります。ルートシードをバックアップファイルと同じ場所に保存することはお勧めしません。一方のファイルセットが漏洩したり盗まれたりした場合、両方が漏洩したり盗まれたりしたことになります。したがって、バックアップデータ用とシードバックアップ用で別々の場所を用意することをお勧めします。

Docker Swarmでのシークレットの暗号化の有効化

Docker Swarmでシークレットの暗号化を有効にするには:

1. まず、**初期シードシークレット**を作成します
2. `blackduck-config.env`を編集して、`SYNOPSISYS_CRYPTO_ENABLED`を次に設定します: `true`
3. Docker Swarm導入の一部として、`docker-compose.encryption.yml`の使用を開始します

Docker Swarm導入コマンドは次のようになります。

```
docker stack deploy -c docker-compose.yml [-c <other compose yaml files> ...] -c docker-compose.encryption.yml blackduck
```

キーシード管理スクリプト

サンプル管理スクリプトは、Black Duck GitHubパブリックリポジトリで確認できます。

<https://github.com/blackducksoftware/secrets-encryption-scripts>

このスクリプトは、Black Duckシークレットの暗号化を管理するためではなく、ここに文書化されている、低レベルのDockerおよびKubernetesコマンドの使用を示すためのものです。2つのスクリプトセットがあり、それぞれが専用のサブディレクトリにあります(Kubernetesプラットフォームでの使用、Docker Swarmプラットフォームでの使用に対応しています)。KubernetesおよびDocker Swarm用の個々のスクリプト間に1対1の対応があります(該当する場合)。たとえば、両方のスクリプトセットに次のスクリプトが含まれています。

`createInitialSeeds.sh`

シードの生成

OpenSSLでのシードの生成

シードの内容は、少なくとも1024バイト長の、セキュリティで保護されたランダムな内容を生成する任意のメカニズムを使用して生成できます。シードは、作成され、シークレットに保存されたら、すぐにファイルシステムから削除し、プライベートな、セキュリティで保護された場所に保存する必要があります。

OpenSSLコマンドは、次のとおりです。

```
openssl rand -hex 1024 > root_seed
```

Docker Swarmでのシードの生成

Docker Swarmでは、シークレットを作成・削除するには、Black Duckを停止する必要があります。Docker Swarmコマンドは、次のとおりです。

```
docker secret create crypto-root-seed ./root_seed
```

Docker Swarmでは、オーケストレーション ファイルで構成されているシークレットが存在する必要があり、長さをゼロにはできません。この制限を回避するために、Black Duckでは、2バイト以下の「プレースホルダ」暗号化シードは、存在しないものとして処理されます。したがって、Docker Swarmで次のコマンドを使用して、前のキー シークレットを削除できます。

```
echo -n "1" | docker secret create crypto-prev-seed -
```

オーケストレーション ファイルで暗号化を有効にしたら、Black Duckシークレットの暗号化を有効にする前に、ここに示すコマンドと同様のコマンドで3つのシード シークレットを作成する必要があります。[サンプル スクリプト](#)を参照してください。

バックアップシードの構成

災害復旧シナリオでシステムを確実に復元できるように、バックアップルートシードを用意することをお勧めします。バックアップルートシードは、システムを復元するために配置できる代替ルートシードです。したがって、ルートシードと同じ方法で安全に保管する必要があります。

バックアップルートシードは、いったんシステムに関連付けられると、ルートシードのローテーションにわたって利用できるという特別な機能がいくつかあります。バックアップシードがシステムによって処理されたら、攻撃および漏洩への暴露を限定するために、シークレットから削除する必要があります。バックアップルートシードは、シークレットがシステム内でどの時点でも「アクティブ」にならないようにする必要があるため、異なる(あまり頻繁ではない)ローテーションスケジュールを持つことができます。

ルートシードをローテーションする必要があるかローテーションしたい場合は、まず、現在のルートシードを前のルートシードとして定義する必要があります。その後、新しいルートシードを生成し、所定の場所に配置できます。

システムがこれらのシードを処理するとき、リソースをローテーションして新しいルートシードを使用するために、前のルートキーが使用されます。この処理の後、前のルートシードをシークレットから削除して、ローテーションを完了し、古いリソースをクリーンアップする必要があります。

バックアップルートシードの作成

バックアップシード/キーは、最初に作成されると、TDEK(テナントの復号化、暗号化キー)低レベルキーをラップします。サンプルスクリプトcreateInitialSeeds.shは、ルートシードとバックアップシードの両方を作成します。Black Duckは、実行されると、両方のキーを使用してTDEKをラップします。

この操作が完了し、ルートシードとバックアップシードの両方が別の場所に安全に保存されたら、バックアップシードシークレットを削除する必要があります。[サンプルスクリプト](#)cleanupBackupSeed.shを参照してください。

ルートキーが紛失または漏洩した場合、バックアップキーを使用してルートキーを置き換えることができます。[サンプルスクリプト](#)useRootSeed.shを参照してください。

バックアップシードのローテーション

ルートキーと同様に、バックアップシードは定期的にローテーションする必要があります。ルートシード(古いルートシードが以前のシードシークレットとして保存され、新しいルートシードシークレットがシステムに提示されます)とは異なり、バックアップシードは新しいバックアップシードを作成するだけでローテーションされます。[サンプルスクリプト](#)rotateBackupSeed.shを参照してください。

ローテーションが完了したら、新しいバックアップ シードを安全に保存し、Black Duckホスト ファイル システムから削除する必要があります。

シークレットローテーションの管理

組織のセキュリティポリシーに従って、使用中のルートシードを定期的にローテーションすることをお勧めします。これを行うには、ローテーションを実行するために追加のシークレットが必要です。ルートシードをローテーションするために、現在のルートシードが「前のルートシード」として構成され、新しく生成されるルートシードがルートシードとして生成および構成されます。システムがこの構成を処理すると（詳細は以下）、シークレットがローテーションされます。

その時点では、古いシードと新しいシードの両方が、システムの内容をアンロックできます。デフォルトでは、新しいルートシードが使用され、システムが意図したとおりに動作していることをテストおよび確認できます。すべてが確認されたら、「前のルートシード」を削除することで、ローテーションを完了します。

前のルートシードは、システムから削除されると、システムの内容のアンロックに使用できなくなるため、破棄してかまいません。これで、新しいルートシードが適切なルートシードになりました。このルートシードは、適切にバックアップおよびセキュリティ保護する必要があります。

ルート キーは、Black Duck のシークレットを実際に暗号化/復号化する、低レベルの TDEK(テナントの復号化、暗号化キー)をラップするために使用されます。定期的に、Black Duck管理者にとって都合が良く、ユーザー組織のルールに準拠しているタイミングで、ルート キーをローテーションする必要があります。

ルートキーをローテーションする手順としては、現在のルートシードの内容で以前のシードシークレットを作成します。その後、新しいルートシードが作成され、ルートシードシークレットに保存される必要があります。

Docker Swarmでのシークレットローテーション

Docker Swarmの場合、Black Duckを停止し、3つのシード操作を実行してから、Black Duckを再び起動する必要があります。ルートシードが、前のシードextractRootAsPreviousSeed.shとして実行中のBlack Duckインスタンスから抽出されます。Docker Swarm[サンプルスクリプト](#)rotateRootSeed.shを参照してください。

ローテーションが完了したら、前のシードシークレットを削除する必要があります。[サンプルスクリプト](#) cleanupPreviousSeed.shを参照してください。Docker Swarmでは、システムをダウンさせ、前のキーを削除してから、Black Duckを再び起動する必要があります。

ローテーションの状態は、ユーザーインターフェイスで、[管理者] > [システム情報] > [暗号]の順に移動し、暗号診断タブを表示することで追跡できます。

Blackduck Storageのカスタムボリュームの構成

ストレージコンテナは、[レポート](#)などファイルベースのオブジェクトを保存するために、最大3個の[複数ボリューム](#)を使用するように構成できます。さらに、この構成は[あるボリュームから別のボリュームにオブジェクトを移行](#)するように設定できます。

複数ボリュームの設定

swarm導入ファイルの中には、docker-compose.storage-overrides.ymlという名前のファイルがあります。このファイルは、カスタマイズが必要な一種のテンプレートです。導入ファイルに含まれていますが、より多くのボリュームを使用できるように、デフォルトのストレージ構成を上書きすることが目的です。

このファイルには、Environment、Storage Volumes、Service Volumesの3つのセクションがあります。

複数のボリュームを使用する理由

デフォルトでは、ストレージコンテナは、単一のボリュームを使用してすべてのオブジェクトを格納します。このボリュームのサイズは、一般的なユーザーが保存オブジェクトに使用する容量に基づいています。使用状況はユーザーごとに異なるため、ボリュームで提供可能な容量よりも多くの空き容量が必要になる場合もあります。すべてのボリュームが拡張可能とは限りません。したがって、別の大きなボリュームを追加して、その新しいボリュームにデータを移行しなければならない場合もあります。

複数のボリュームが必要になるもう1つの理由は、ボリュームがリモートシステム（NASまたはSAN）でホストされており、そのリモートシステムが廃止される予定になった場合です。ホストする2番目のボリュームを新しいシステムで作成し、コンテンツをそこに移動する必要があります。

Environment

Environmentセクションは、次のように構成されています。

```
services:
  storage:
    environment:
      # Provider 1 settings
      BLACKDUCK_STORAGE_PROVIDER_ENABLED_1: 'true'
      BLACKDUCK_STORAGE_PROVIDER_PREFERENCE_1: 10
      BLACKDUCK_STORAGE_PROVIDER_READONLY_1: 'false'
      BLACKDUCK_STORAGE_PROVIDER_MIGRATION_MODE_1: 'none'
      # Provider 2 settings
      BLACKDUCK_STORAGE_PROVIDER_ENABLED_2: 'false'
      BLACKDUCK_STORAGE_PROVIDER_PREFERENCE_2: 20
      BLACKDUCK_STORAGE_PROVIDER_READONLY_2: 'false'
      BLACKDUCK_STORAGE_PROVIDER_MIGRATION_MODE_2: 'none'
      # Provider 3 settings
      BLACKDUCK_STORAGE_PROVIDER_ENABLED_3: 'false'
      BLACKDUCK_STORAGE_PROVIDER_PREFERENCE_3: 30
      BLACKDUCK_STORAGE_PROVIDER_READONLY_3: 'false'
      BLACKDUCK_STORAGE_PROVIDER_MIGRATION_MODE_3: 'none'
      ...
```

これらの設定により、3つの各ストレージプロバイダが有効または無効になります。ファイル内の設定は、デフォルトと同じ設定で出荷されます（プロバイダ1が有効、2および3が無効）。

プロバイダを設定するには、指定したプロバイダの設定を変更します。設定は次のとおりです。

設定	詳細
BLACKDUCK_STORAGE_PROVIDER_ENABLED_(N)	<p>デフォルト: プロバイダ1の場合はtrue、その他の場合は false</p> <p>有効な値: trueまたは false</p> <p>注記: これにより、プロバイダが有効または無効になります。1つ以上のプロバイダを有効にする必要があります。有効にしないと、システムを起動できません。</p>
BLACKDUCK_STORAGE_PROVIDER_PREFERENCE_(N)	<p>デフォルト: プロバイダ数の10倍</p> <p>有効な値: 0-999</p> <p>注記: この値は、プロバイダ間の優先度を示します。新しいコンテンツを保存する場合、優先度が最大（優先度の数値は最小）のプロバイダが使用されます。構成内の他のプロバイダでは、引き続きコンテンツおよびその他の全機能が提供されますが、新しいコンテンツは追加されません。</p> <p>注: アクティブな全プロバイダには、固有な優先度の数値が必要です。2つのプロバイダが同じ値になることは許されません。</p>
BLACKDUCK_STORAGE_PROVIDER_READONLY_(N)	<p>デフォルト: false</p> <p>有効な値: trueまたは false</p> <p>注記: これはプロバイダが読み取り専用であることを示します。優先度が最大（優先度の数値は最小）のプロバイダは読み取り専用にできません。読み取り専用にすると、システムが機能しなくなります。</p>

設定	詳細
	読み取り専用プロバイダでは、データの追加やデータの削除によってストレージボリュームが変更されることはありません。ただし、データベース内のメタデータは、オブジェクトの削除やその他の変更を記録するように制御されます。
BLACKDUCK_STORAGE_PROVIDER_MIGRATION_MODE_(N)	デフォルト: none 有効な値: none、drain、delete、duplicate 注記: この値はプロバイダの移行モードを設定します。このモードの詳細と使い方については、このドキュメントの移行セクションを参照してください。

Storage volumes

このセクションでは、ソフトウェアが想定すべきシステム上の特定マウントポイントを、指定したストレージボリュームに割り当てます。プロバイダ1のボリュームは、「storage-volume」という名前で、別の場所で定義されます。ストレージプロバイダ1が有効になっていない場合でも、プロバイダ1は常に存在しています。上書きファイルのStorage volumesセクションは、次のようになります。

```
services:
  storage:
    environment:
      ...
    volumes:
      ## NOTE: File provider 1's volume mount point is always
      ## present since it is defined as the upstream default
      ## It will only be used if the file provider 1 is enabled
      ##
      ## File provider 2's volume mount point
      #- storage-volume2:/opt/blackduck/hub/uploads2
      ##
      ## File provider 3's volume mount point
      #- storage-volume3:/opt/blackduck/hub/uploads3
      ...
```

他のプロバイダの場合は、有効にしたプロバイダごとに、当該セクションをコメント解除するだけです。例えば、プロバイダ2が有効になっている場合は、ファイルのこのセクションは次のようになります。

```
...
  ##
  ## File provider 2's volume mount point
  - storage-volume2:/opt/blackduck/hub/uploads2
  ##
  ...
```


Service volumes

このセクションでは、名前とその構成によってボリュームを定義します。提供される構成では、Docker Swarmのデフォルトボリュームドライバを使用します。Service volumesセクションと同様に、有効になっているプロバイダに基づいて、適切な部分をコメント解除するだけです。ファイルのService Volumeセクションは次のようになります。

```
services:
  storage:
    environment:
      ...
    volumes:
      ...
    volumes: {
      ## NOTE: File provider 1's storage volume is always present
      ## since it is defined as the upstream default. It will
      ## only be used if the file provider 1 is enabled
```



```
##
## File provider 2's storage volume
#storage-volume2: null,
##
## File provider 3's storage volume
#storage-volume3: null,
}
```

 注：ここでは、デフォルトストレージドライバ以外の設定を変更できます。「storage-volume2」という名前のストレージボリュームを、NFS、NAS、またはSANIによって作成できます。これらの設定は、ベンダーの設定や拡張機能とともに、お客様の環境を考慮したうえで、標準のDocker Swarmパラメータで指定する必要があります。したがって、ここでは十分に説明することができません。

ボリューム間の移行

複数のボリュームを設定した場合、1個以上のプロバイダボリュームから新しいプロバイダボリュームにコンテンツを移行できます。これは、優先度が最高（優先度の数値が最小）ではないプロバイダに対してのみ実行できます。この操作を実行するには、次のいずれかの移行モードでボリュームを設定します。設定後、移行を開始するために、Black Duckを再起動する必要があります。移行は、完了するまで、ジョブによりバックグラウンドで実行されます。

移行モード	詳細
none	目的: 移行が進行中でないことを示します。 注記: デフォルトの移行モード。
drain	目的: このモードでは、設定されているプロバイダから、優先度が最大（優先度の数値は最小）のプロバイダにコンテンツが移動されます。コンテンツが移動されると、コンテンツはすぐにソースプロバイダから削除されます。 注記: これは、ターゲットプロバイダに追加し、ソースから削除するという直接移動の操作です。
delete	目的: このモードでは、設定されているプロバイダから、優先度が最大（優先度の数値は最小）のプロバイダにコンテンツがコピーされます。コンテンツがコピーされると、ソースプロバイダでは削除対象のマークがコンテンツに付けられます。標準的な削除保留期間が適用されます。この期間が経過すると、コンテンツは削除されます。 注記: この移動の場合、削除の保留期間中、ソースプロバイダのコンテンツは存続可能な状態で保持されており、バックアップからシステムをリカバリできるようになっています。デフォルトの削除保留期間は6時間です。
duplicate	目的: このモードでは、設定されているプロバイダから、優先度が最大（優先度の数値は最小）のプロバイダにコンテンツがコピーされます。コンテンツがコピーされても、メタデータを含めて、ソースのコンテンツは変更されません。 注記: 複製移行の後、データベース内の全コンテンツとメタデータが保存された2つのボリュームが存在することになります。「複製とダンプ」プロセスで次の手順に進み、元のボリュームの構成を解除した場合、コンテンツファイルは削除されますが、メタデータはデータベース内に残されたままになります。この場合、不明なボリュームを参照すると、ブルーニングジョブで警告が生


移行モード	詳細
	<p>成されます(ジョブエラー)。このエラーを解決するには、次のプロパティを使用して、孤立したメタデータレコードのプルーニングを有効にします。</p> <pre>storage.pruner.orphaned.data.pruning.enable=true</pre>


レポートのストレージボリュームの設定

ストレージボリュームのレポートプロパティは、blackduck-config.envなどのシステムプロパティで、以下のプロパティを使用して設定できます：

- blackduck.hub.maximum.report.age: Black Duck 永続ストレージ ボリュームに保管されるレポートの最大経過時間を設定します。この値(日数)を上回る期間、経過しているレポートを削除します。デフォルト値は、180日です。
このプロパティの最大推奨値は365日です。
- blackduck.hub.maximum.reports.per.user: 各ユーザーがBlack Duckで作成できるレポートの最大数を設定します。この値を超えると、このユーザーが生成したレポートは、このユーザーによって生成された最も古いレポートから削除されます。デフォルト値は、100レポートです。

このプロパティの最大推奨値は1000レポートです。この値については、100人未満のユーザーが数多くのレポートを作成することを想定しています。合計レポート数が100,000を超えることは推奨されていません。

 注：Black Duck の試算では、Black Duckのレポート25,000件に必要な追加のディスク ストレージは約3 GBで、レポート数の増加に併せて定量的に増加します。管理者は、ディスク使用量を監視し、十分な追加容量を確保することを推奨します。

 警告：Black Duckに10万個以上のレポートを長期間保管することはお勧めできません。長期の保管あるいは監査用ストレージとしては、Black Duckシステム外の外部データ ストレージが推奨されます。この推奨値を超えると、レポート生成、API、一般的なBlack Duckデータベースの性能が低下する可能性があります。

jobrunnerスレッドプールの設定

Black Duck には、2 つのジョブ プールがあります。1 つはスケジュールされたジョブを実行するプール(別名: 定期プール)、もう 1 つは API やユーザーによる操作など、特定のイベントで開始されるジョブを実行するプール(別名: オンデマンド プール)です。

各プールには、最大スレッドとプリフェッチの2つの設定があります。

[最大スレッド]は、jobrunnerコンテナが同時に実行できるジョブの最大数です。ほとんどのジョブはデータベースを使用しますが、最大32の接続を確立できるため、定期的な最大スレッドとオンデマンドの最大スレッドの合計数が32を超えないように注意してください。jobrunnerメモリは使用率がすぐに最大になるため、デフォルトのスレッド数は非常に小さい値に設定されています。

[プリフェッチ]は、データベースへの1回のアクセスで取得するjobrunnerコンテナあたりのジョブ数です。大きな値を設定すると効率的ですが、小さい値を設定すると、複数のjobrunner間で負荷がより均等に分散されます(一般的には、負荷分散もjobrunnerの設計目標ではありません)。

Docker Swarmの場合、blackduck-config.envファイルに次の設定を追加します。

```
org.quartz.scheduler.periodic.maxthreads=2
org.quartz.scheduler.periodic.prefetch=1
org.quartz.scheduler.ondemand.maxthreads=4
org.quartz.scheduler.ondemand.prefetch=2
```


構成表サポートのための高速スキャンの設定

構成表のサポートのために、高速スキャンを設定して、データポイントを含めた完全な結果形式を提供できます。これを実行するには、次の環境変数を設定します。BLACKDUCK_RAPID_SCAN_EXTENDED_DATA=true.

長時間実行しているジョブのしきい値を変更

以下の変数をblackduck-config.envファイルへ追加することで、長時間実行しているジョブを判定するしきい値を構成できます。

- BLACKDUCK_DEFAULT_JOB_RUNTIME_THRESHOLD_HOURS=[時間単位の値]

この環境変数のデフォルト値は24時間です。

SCM統合の有効化

この機能はデフォルトではBlack Duckで有効になっていないため、アクティブ化する必要があります。アクティブ化するには、この機能をお使いの製品登録キーへ追加し、以下をvalues.yamlファイルに追加してください。

```
enableIntegration: true
```

 注：Black Duck は、現時点ではSCM統合の自己署名証明書を受け入れません。

自動スキャンRetryヘッダーの設定

自動スキャン再試行ヘッダーは、ネットワークの不安定、サービスの中断、またはスキャンを妨害する可能性のあるその他の問題など、一時的な問題が発生した場合でも、スキャンが正常に完了するようサポートします。キューイングシステムを使用してスキャンを自動で再試行することで、システムは手動介入の必要性を減らせます。自動スキャン再試行ヘッダーを設定することで、管理者は再試行の動作を特定のニーズや環境に合わせて調整できます。このカスタマイズにより、再試行プロセスが最適化され、システムのパフォーマンスと運用要件に確実に一致するようになります。

blackduck-config.envファイルに以下のプロパティを追加することにより、429レスポンスのRetry-Afterヘッダーを変更して、Detectにレート制限スキャンを再試行するタイミングを知らせることができます：

- BLACKDUCK_USE_QUEUE_RATE_LIMITING:true を設定し、ご利用の環境下でキューベースのレート制限を有効にします。デフォルト値は、falseです。
- BLACKDUCK_INITIAL_RATE_LIMIT_DURATION_BRACKET_THRESHOLD_MINUTES: システムが最初の再試行期間から次の再試行期間へ移行する前に、システムがレート制限を課す必要がある期間を指示します。デフォルト値は、2分間です。
- BLACKDUCK_RATE_LIMIT_DURATION_THRESHOLD_BRACKET_INCREMENT_MINUTES: システムが次の再試行期間と乗数へ移行する前に、レート制限ブラケットに留まる期間と乗数を指示します。デフォルト値は、2分間です。
- BLACKDUCK_INITIAL_RETRY_DURATION_MINUTES: 最初のレート制限ブラケットにおける、Retry-Afterヘッダーの最初の期間。デフォルト値は、1分間です。
- BLACKDUCK_RETRY_DURATION_MULTIPLIER_MINUTES: システムが新しいレート制限期間ブラケットへ達するたびにRetry-After期間に掛ける乗数。デフォルト値は、2分間です。

```
BLACKDUCK_USE_QUEUE_RATE_LIMITING=TRUE
BLACKDUCK_INITIAL_RATE_LIMIT_DURATION_BRACKET_THRESHOLD_MINUTES=2
```

4. 管理タスク・階層的サブプロジェクトのライセンス競合設定

```
BLACKDUCK_RATE_LIMIT_DURATION_THRESHOLD_BRACKET_INCREMENT_MINUTES=2
BLACKDUCK_INITIAL_RETRY_DURATION_MINUTES=1
RETRY_DURATION_MULTIPLIER_MINUTES_KEY=2
```

階層的サブプロジェクトのライセンス競合設定

デフォルトでは、階層的サブプロジェクトのライセンス競合は、ご利用の環境で有効化されています。以下のパラメータを設定することにより、階層的サブプロジェクトのライセンス競合を無効にできます：

```
USE_HIERARCHICAL_LICENSE_CONFLICTS=FALSE
```

サブプロジェクトの深さは、デフォルトで5レベルまで設定できますが、以下のパラメータで設定できます：

```
HIERARCHICAL_LICENSE_CONFLICT_DEPTH_LIMIT=<value desired>
```

JWT 公開/秘密鍵ペアのプロビジョニング

JWT 管理のセキュリティと柔軟性を強化するため、当社のシステムは公開/秘密鍵ペアのオプション プロビジョニングをサポートするようになりました。これによってこれらのキーを安全に提供・管理できるため、秘密鍵の認証サービスや公開鍵のパブリック API サービスなど、適切なサービスのみでキーが使用されるようになります。

現在、RSA キー（PEM エンコード済み）のみがサポートされています。具体的には、公開鍵は X.509 形式、秘密鍵は PKCS#8 形式である必要があります。

Docker シークレットの作成

Docker でパブリック/プライベート シークレットを作成するには、次の手順に従います。

1. 次のコマンドを入力します。

```
docker secret create hub_JWT_PUBLIC_KEY public-key.pem
docker secret create hub_JWT_PRIVATE_KEY private-key.pem
```

2. JWT シークレットを使用するように docker-compose.local-overrides.yml を編集し、デプロイします。

```
docker stack deploy -c docker-compose.yml -c docker-compose.local-overrides.yml jwt-swarm
```

サンプル オーバーライド ファイル

以下にサンプル docker-compose.local-overrides.yml ファイル（必要に応じて設定された統合サービス）を示します。このファイルのコメントは、最も一般的なオプション セットの一部をオーバーライドする方法を示しています。ただし、ここにオーバーライドを追加することで、ポート マッピングなどの Docker 構成設定をオーバーライドできます。

```
version: '3.6'
services:
  authentication:
    secrets:
      - JWT_PUBLIC_KEY
      - JWT_PRIVATE_KEY
  webapp:
    secrets:
      - JWT_PUBLIC_KEY
      - JWT_PRIVATE_KEY
  scan:
    secrets:
      - JWT_PUBLIC_KEY
      - JWT_PRIVATE_KEY
  storage:
```

```

  secrets:
    - JWT_PUBLIC_KEY
    - JWT_PRIVATE_KEY
jobrunner:
  secrets:
    - JWT_PUBLIC_KEY
    - JWT_PRIVATE_KEY
bomengine:
  secrets:
    - JWT_PUBLIC_KEY
    - JWT_PRIVATE_KEY
matchengine:
  secrets:
    - JWT_PUBLIC_KEY
    - JWT_PRIVATE_KEY
#integration:
#  secrets:
#    - JWT_PUBLIC_KEY
#    - JWT_PRIVATE_KEY
secrets:
  JWT_PUBLIC_KEY:
    external: true
    name: "hub_JWT_PUBLIC_KEY"
  JWT_PRIVATE_KEY:
    external: true
    name: "hub_JWT_PRIVATE_KEY"

```

セッショントークンの無効化を設定

セッショントークンの無効化をアクティブ化するには、`blackduck-config.env` ファイルを設定して `JWT_BLOCK_LIST_CHECK=true` を設定します。

5. アンインストール: Black Duck


Black Duckをアンインストールするには、次の手順に従います:

- ・ コンテナ、ネットワーク、シークレットを停止して削除します。

```
docker stack rm ${stack name}
```

- ・ すべての未使用のボリュームを削除します:


```
docker volume prune -a
```


 **注意:** このコマンドは、未使用のボリュームをすべて削除します。どのコンテナからも参照されていないボリュームが削除されます。これには、他のアプリケーションで使用されていない未使用のボリュームが含まれます。

PostgreSQLデータベースはバックアップされません。[データベースをバックアップする](#)には、これらの手順を使用します。


6. アップグレード Black Duck

Black Duck は、利用可能な任意のバージョンへのアップグレードをサポートしているため、1回のアップグレードで複数のバージョンを飛び越えることができます。

 注：データベース管理者は、2023.4.0以降をインストールまたは 2023.4.0以降にアップグレードする前に、hstore PostgreSQL拡張機能のインストールを有効にする必要があります。

 注：2019.8.0より前のバージョンからアップグレードする場合、VulnerabilityReprioritizationJobとVulnerabilitySummaryFetchJobの2つのジョブが起動時に実行され、脆弱性データが同期されます。

これらのジョブの実行には時間がかかる場合があり、既存の構成表の全体的な脆弱性スコアは、これらのジョブが完了するまで使用できません。システム管理者の役割を持っているユーザーは、Black Duckの[ジョブ]ページを使用してこれらのジョブを監視できます。

 注：2018.12.0より前のバージョンからアップグレードする場合は、このリリースの新機能をサポートするためにデータ移行が必要になるため、通常よりもアップグレードに時間がかかります。アップグレード時間は、Black Duckデータベースのサイズによって異なります。アップグレードプロセスを監視する場合は、Black Duckカスタマー サポートにお問い合わせください。

インストールファイル

インストールファイルはGitHubで入手できます。

オーケストレーションファイルをダウンロードします。インストール/アップグレードプロセスの一環として、これらのオーケストレーションファイルは必要なDockerイメージをプルダウンします。

tar.gzファイルのファイル名はアクセス方法によって異なりますが、内容は同じです。

GitHubページからダウンロードする場合

1. リンクを選択して、GitHubページからtar.gzファイルをダウンロードします：<https://github.com/blackducksoftware/hub>。
2. Black Duck .gzファイルを解凍します。
`gunzip hub-2024.10.0.tar.gz`
3. Black Duck.tarファイルを展開します。
`tar xvf hub-2024.10.0.tar`

wgetコマンドを使用してダウンロードする場合


1. 次のコマンドを実行します。
`wget https://github.com/blackducksoftware/hub/archive/v2024.10.0.tar.gz`
2. Black Duck .gzファイルを解凍します。
`gunzip v2024.10.0.tar.gz`
3. Black Duck.tarファイルを展開します。
`tar xvf v2024.10.0.tar`

監査イベントテーブルの未使用の行をパージする移行スクリプト

このセクションは、2019.12.0より前のバージョンのBlack Duckからアップグレードする場合にのみ適用されます。

アップグレード中に、移行スクリプトが実行されて、レポートデータベースの変更によりaudit_eventテーブルで使用されなくなった行がパージされます。audit_eventテーブルのサイズによっては、このスクリプトの実行に時間がかかる場合があります。たとえば、350 GBのaudit_eventテーブルの場合、移行スクリプトの実行には約20分かかります。


監査イベントテーブルのサイズを判断するには、次のいずれかのタスクを実行します。


- ・ bds_hubデータベースで、次のコマンドを実行します。
SELECT pg_size_pretty(pg_total_relation_size('st.audit_event'));
- ・ Black Duck UIにシステム管理者としてログインし、次の手順を実行します。
 1. 展開メニューアイコン()をクリックし、[管理]を選択します。
 2. [管理]ページで[システム情報]を選択します。
[システム情報]ページが表示されます。
 3. ページの左側の列で[db]を選択します。
 4. テーブルサイズテーブルで、audit_eventテーブル名のtotal_tbl_size値を見つけます。

schemaname	tablename	total_tbl_size_pretty	tbl_size_pretty	total_tbl_size	tbl_size
st	scan_composite_leaf	6508 MB	4232 MB	6823731200	4437254144
st	audit_event	2027 MB	1577 MB	2125168640	1653915648

アップグレードが完了したら、audit_eventテーブルでVACUUM FULLコマンドを実行して、PostgreSQLのパフォーマンスを最適化することを強くお勧めします。

- ・ システムの使用状況によっては、VACUUM FULLコマンドを実行することで、Black Duckによって使用されなくなった大量のディスク容量を再利用できる可能性があります。
- ・ このコマンドを実行すると、クエリのパフォーマンスが向上します。

 注： VACUUM FULLコマンドを実行しないと、パフォーマンスが低下する可能性があります。

 重要： VACUUM FULLコマンドを実行するのに十分な容量があることを確認する必要があります。そうしないと、ディスク容量が不足して失敗し、データベース全体が破損する可能性があります。VACUUM FULLコマンドは、audit_eventテーブルで現在使用されているディスク容量の2倍の容量を必要とします。

コンテナ化されたPostgreSQLデータベースの導入でVACUUM FULLコマンドを実行するには、次の手順を実行します。

1. audit_eventテーブルのサイズを取得し、VACUUM FULLコマンドを実行するのに十分な容量があることを確認します。
2. docker psコマンドを実行して、PostgreSQLコンテナのIDを取得します。
3. 次のコマンドを実行して、PostgreSQLコンテナにアクセスします。
docker exec -it <container_ID> psql bds_hub
4. 次のVACUUM FULLコマンドを実行して、使用されなくなった容量を再利用します。
VACUUM FULL ANALYZE st.audit_event;

外部PostgreSQLデータベースを導入している場合は、audit_eventテーブルのサイズを判断し、VACUUM FULLコマンドを実行し、完了したら導入を再起動する必要があります。

既存のDockerアーキテクチャからのアップグレード

以前のバージョンのBlack Duckからアップグレードするには、次の手順を実行します。

1. PostgreSQLデータベースを移行します。
2. Black Duckのアップグレード。

PostgreSQLデータベースの移行

Black Duck 2022.10.0はPostgreSQLイメージをバージョン11からバージョン13に移行しており、PostgreSQL 9.6コンテナ(バージョン4.2から2021.10.xまで)またはPostgreSQL 11コンテナ(バージョン2022.2.0から2022.7.xまで)を使用したバージョンからのアップグレードをサポートしています。インストール中に、blackduck-postgres-upgraderコンテナは既存のデータベースをPostgreSQL 13に移行し、完了すると終了します。

次の点に注意してください。

- ・ コア以外のPG拡張機能を使用しているお客様の場合は、移行前にそれらをアンインストールし、移行が正常に完了した後に再インストールすることを強くお勧めします。そうしないと、移行が失敗する可能性があります。
- ・ レプリケーションを設定しているお客様は、移行前に、pg_upgradeのドキュメントの手順に従う必要があります。そこで説明されている準備が行われていない場合、移行はおそらく成功しますが、レプリケーションの設定が壊れます。
- ・ Black Duckが提供するPostgreSQLイメージを使用していないお客様は、サポートされているバージョンを使用していることを確認する必要があります。Black Duck 2022.10.0はPostgreSQLバージョン13.xおよび14.xをサポートしており、Black Duckでは最新リリース14.xの使用を推奨しています。
 - ・ Community PostgreSQLのユーザーは、PostgreSQLのアップグレードについて、PostgreSQL 14の手順(<https://www.postgresql.org/docs/14/pgupgrade.html>)に従う必要があります。
 - ・ サードパーティのPostgreSQL(RDSなど)のユーザーは、プロバイダの指示に従う必要があります。
- ・ 内部PostgreSQLコンテナのユーザーであるPostgreSQLは、必要に応じて自動的にアップグレードされます。
- ・ Black Duck 4.2.0より前のBlack Duckバージョンのユーザーは、アップグレードする前にBlack Duckのテクニカルサポートにお問い合わせください。

❗ 重要：移行を開始する前に：

- ・ システムカタログのデータコピーによるディスクの使用に起因する予期しない問題を回避するため、10%ほどの余裕をディスク容量に確保してください。
- ・ ディスク容量が不足するとLinuxシステムが中断する可能性があるため、ルートディレクトリの容量とボリュームマウントを確認してください。

移行は完全に自動化されているため、Black Duckの標準アップグレードの操作以外に追加の操作は必要ありません。上記のレイアウトとUIDの変更を行うには、blackduck-postgres-upgraderコンテナをルートとして実行する必要があります。

その後のBlack Duckの再起動時に、blackduck-postgres-upgraderは移行が不要であると判断し、すぐに終了します。

アップグレード Black Duck

Black Duckをアップグレードするには：

1. 次のいずれかを実行します。

6. アップグレード Black Duck・既存のDockerアーキテクチャからのアップグレード

- ・ docker-compose.local-overrides.ymlを使用して.ymlファイルを変更しなかった場合は、新しいバージョンのBlack Duckのdocker-swarmディレクトリにあるファイルを使用して、次のいずれかのコマンドを実行します。

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml hub
```

(DBコンテナを使用している場合)

- ・

```
docker stack deploy -c docker-compose.externaldb.yml -c sizes-gen04/120sph.yaml hub
```

(外部PostgreSQLインスタンスを使用している場合)

- ・ Black Duck Binary Analysisをアップグレードする場合は、新しいバージョンのBlack Duckのdocker-swarmディレクトリにあるファイルを使用して、次のコマンドのいずれかを実行します。

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.bdba.yml hub
```

(DBコンテナとBlack Duck Binary Analysisを使用している場合)

- ・

```
docker stack deploy -c docker-compose.externaldb.yml -c sizes-gen04/120sph.yaml -c docker-compose.bdba.yml hub
```

(外部データベースとBlack Duck Binary Analysisを使用している場合)

- ・ docker-compose.local-overrides.ymlを使用して.ymlファイルを変更した場合は、次のコマンドのいずれかを実行します。変更を含むバージョンのdocker-compose.local-overrides.ymlファイルを使用します。他のすべてのファイルについては、新しいバージョンのBlack Duckのdocker-swarmディレクトリにあるファイルを使用します。

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-overrides.yml hub
```

(DBコンテナを使用している場合)

- ・

```
docker stack deploy -c docker-compose.externaldb.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-overrides.yml hub
```

(外部PostgreSQLインスタンスを使用している場合)

- ・

```
docker stack deploy -c docker-compose.yml -c docker-compose.bdba.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-overrides.yml hub
```

(DBコンテナとBlack Duck Binary Analysisを使用している場合)

- ・



```
docker stack deploy -c docker-compose.externaldb.yml -c docker-compose.bdba.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-overrides.yml hub
```

(外部データベースとBlack Duck Binary Analysisを使用している場合)

- ・ Swarmサービスに対してファイルシステムが読み取り専用の状態でBlack Duckをアップグレードするには、docker-compose.readonly.ymlファイルを前の例に追加します。

たとえば、docker-compose.local-overrides.ymlを使用して.ymlファイルを変更し、DBコンテナを使用するBlack Duckインストールをアップグレードする場合は、次のコマンドを実行します。

```
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.readonly.yml -c docker-compose.local-overrides.yml hub
```

-  注：上記のコマンドで使用したリソース定義ファイルは、スキャンパターンと使用法に合わせて変更できます。使用可能なすべてのオプションのリストについては、[配布セクション](#)を参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。
-  注：以前にhub-proxy.envファイルを編集した場合、それらの編集内容をblackduck-config.envファイルにコピーする必要があります。

データのバックアップと復元

データベースが内部の場合（つまり、バックエンドデータベースとしてpostgresコンテナを使用している場合）は、バックアップと復元に、docker-swarm/binフォルダの下にあるスクリプトを使用することができます。

データのバックアップ

1. HUBをデータベース移行モードとして起動します。

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

2. 次のコマンドを実行します。

```
./hub_create_data_dump.sh /pathtodbdump
```

スクリプトが完了すると、/pathtodbdump の下に次のファイルができます。

- bds_hub.dump
- globals.sql

データの復元

データベースを復元する前に、既存のhub_postgres96-data-volumeを削除する必要があります。以下のコマンドを使用して実行します：

```
STACK=${STACK:-hub} -- change the "hub" part to match your actual deployment
docker volume rm ${STACK}_postgres96-data-volume
```

既存のボリュームを削除した後、以下の手順でデータベースの復元を実行できます。

1. データを復元するには、新しいHUBをデータベース移行モードとして起動してから、hub_db_migrate.shを使用してデータを復元します：

```
docker stack deploy -c docker-compose.dbmigrate.yml hub
./hub_db_migrate.sh /pathtodbdump
```

2. 最後に、HUBを通常どおりに起動します：

```
docker stack rm hub
docker stack deploy -c docker-compose.yml -c sizes-gen04/120sph.yaml -c docker-compose.local-
overrides.yml hub
```

上記で使用したリソース定義ファイルは、スキャンパターンと使用法に合わせて変更できます。利用可能なすべてのオプションのリストについては、[配布セクション](#)を参照してください。リソース定義が大きくなるに従ってシステム要件も高度になるため、各オプションの[システム要件](#)に注意してください。

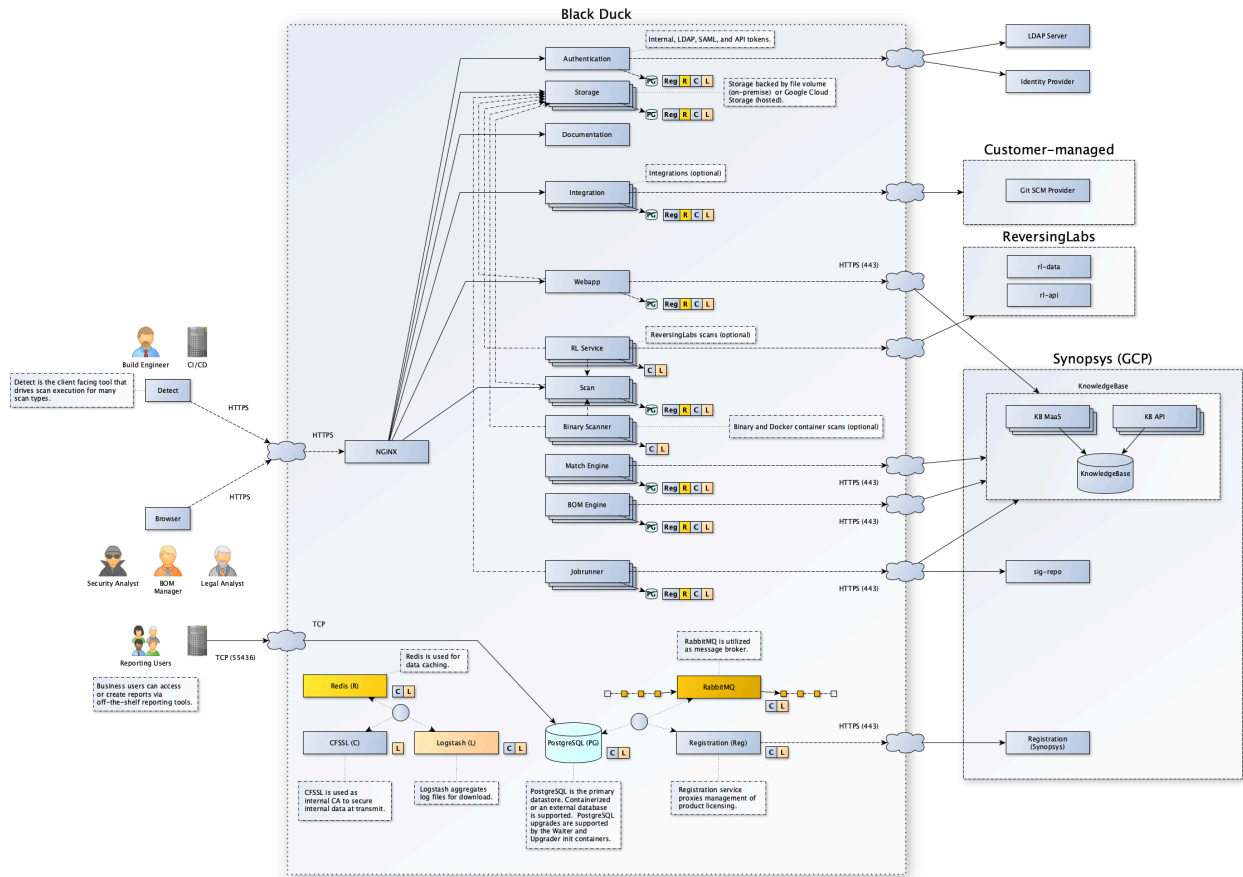
データベースが外部の場合は、データのバックアップについてデータベース管理者にお問い合わせください。

7. Dockerコンテナ

Black Duckアプリケーションを構成するDockerネットワーク内のコンテナを以下に示します。

1. [Authentication](#)
2. [バイナリ スキャナ](#) — Black Duck Binary Analysisが有効になっている場合は必須です。
3. [Bomengine](#)
4. [CA](#)
5. [DB](#) — 外部のPostgreSQLインスタンスを使用する場合、このコンテナはBlack Duckアプリケーションに含まれません。
6. [Documentation](#)
7. [Integration](#)
8. [Jobrunner](#)
9. [Logstash](#)
10. [Matchengine](#)
11. [Rabbitmq](#)
12. [Redis](#)
13. [Registration](#)
14. [RLサービス](#) — ReversingLabsスキャンが有効になっている場合は必須です。
15. [スキャン](#)
16. [ストレージ](#)
17. [Webapp](#)
18. [Webserver](#)

次の図は、コンテナ間の基本的な関係と、Dockerネットワークの外部に公開されるポートを示しています。



ZookeeperコンテナはBlack Duckバージョン2020.4.0で削除されました。次のzookeeperボリュームは使用されなくなったため、手動で削除できます。

- zookeeper-data-volume
- zookeeper-datalog-volume

次の表に、各コンテナに関する詳細情報を示します。

Authenticationコンテナ

コンテナ名 : blackduck-authentication	
イメージ名	blackducksoftware/blackduck-authentication:2024.10.0
説明	authenticationサービスは、すべての認証関連リクエストの宛先となるコンテナです。
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。現在、拡張することはできません。
リンク/ポート	<p>外部に公開されているものではありません(8443は内部に公開)。このコンテナは、次の他のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> • postgres • cfssl • logstash • registration

コンテナ名: blackduck-authentication	
	<ul style="list-style-type: none"> webapp <p>コンテナは、それにリンクする他のコンテナに8443を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> postgres - \$HUB_POSTGRES_HOST cfssl - \$HUB_CFSSL_HOST logstash - \$HUB_LOGSTASH_HOST registration - \$HUB_REGISTRATION_HOST webapp - \$HUB_WEBAPP_HOST
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 512 MB コンテナメモリ: 1 GB コンテナCPU: 1 CPU
ユーザー/グループ	<p>このコンテナはUID 100として実行されます。コンテナがUID 0(root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Binaryscannerコンテナ

次のコンテナは、Black Duck Binary Analysisがある場合にのみインストールされます。

コンテナ名: bdba-worker	
イメージ名	イメージ: blackducksoftware/bdba-worker:2023.03
説明	<p>このコンテナはバイナリファイルを解析します。</p> <p>現在、このコンテナはBlack Duck — Binary Analysisが有効な場合にのみ使用されます。</p>
スケーラビリティ	このコンテナはスケーリングできます。
リンク/ポート	<p>このコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> cfssl logstash rabbitmq webserver <p>コンテナは、それにリンクする他のコンテナにポート5671を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> cfssl: \$HUB_CFSSL_HOST logstash: \$HUB_LOGSTASH_HOST rabbitmq: \$RABBIT_MQ_HOST webserver: \$HUB_WEBSERVER_HOST

コンテナ名: bdba-worker	
リソース/制約	<ul style="list-style-type: none"> ・ デフォルトの最大Javaヒープサイズ: 該当なし ・ コンテナメモリ: 2 GB ・ コンテナCPU: 1 CPU
ユーザー/グループ	このコンテナはUID 0として実行されます。
環境ファイル	hub-bdba.env

Bomengineコンテナ


コンテナ名: bomengine	
イメージ名	blackducksoftware/blackduck-bomengine:2024.10.0
説明	bomengineコンテナは、構成表を作成し、それらを最新の状態に保ちます。
スケーラビリティ	コンテナはスケーリングできます
リンク/ポート	<p>bomengineコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> ・ postgres ・ registration ・ logstash ・ cfssl
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、個々のサービス名が異なる場合があります。たとえば、別のサービス名を使用して解決された外部PostgreSQLエンドポイントがあるとします。このようなユースケースの場合は、これらの環境変数を設定して、デフォルトのホスト名を上書きすることができます。</p> <ul style="list-style-type: none"> ・ postgres: \$HUB_POSTGRES_HOST ・ registration: \$HUB_REGISTRATION_HOST ・ logstash: \$HUB_LOGSTASH_HOST ・ cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> ・ デフォルトの最大Javaヒープサイズ: 4 GB ・ コンテナメモリ: 4.5 GB
ユーザー/グループ	<p>このコンテナはUID 100として実行されます</p> <p>コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

CAコンテナ

コンテナ名: blackduck-cfssl	
イメージ名	blackducksoftware/blackduck-cfssl:1.0.2

コンテナ名: blackduck-cfssl	
説明	このコンテナは、CFSSLを使用し、これはPostgreSQL、NGiNX、およびPostgreに対して認証する必要があるクライアントの証明書生成に使用されます。このコンテナは、アプリケーションを構成する内部コンテナのTLS証明書を生成するのにも使用されます。
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。スケーリングしてはいけません。
リンク/ポート	コンテナは、Dockerネットワーク内で、それにリンクする他のコンテナ/サービスにポート8888を公開する必要があります。
リソース/制約	<ul style="list-style-type: none"> ・ デフォルトの最大Javaヒープサイズ: 該当なし ・ コンテナメモリ: 512 MB ・ コンテナCPU: 未指定
ユーザー/グループ	このコンテナはUID 100として実行されます。コンテナがUID 0(root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。
環境ファイル	blackduck-config.env

DBコンテナ

 注: 外部Postgreインスタンスを使用する場合、このコンテナはBlack Duckアプリケーションに含まれません。

コンテナ名: blackduck-postgres	
イメージ名	blackducksoftware/blackduck-postgres:9.6-1.1
説明	<p>DBコンテナには、オープンソースのオブジェクトリレーショナルデータベースシステムであるPostgreSQLデータベースが格納されます。アプリケーションはPostgreSQLデータベースを使用してデータを格納します。</p> <p>このコンテナには1つのインスタンスがあります。これは、アプリケーションのすべてのデータが格納される場所です。Postgreのポートは2つあります。1つのポートはDockerネットワーク内のコンテナに公開されます。これはアプリケーションが使用する接続です。このポートは証明書認証によって保護されています。もう1つのポートはDockerネットワークの外部に公開されます。これにより、読み取り専用ユーザーは、hub_reportdb_changepassword.shスクリプトを使用して設定されたパスワードを介して接続できます。このポートとユーザーは、レポートとデータ抽出に使用できます。</p> <p>レポートデータベースの詳細については、『レポートデータベース』ガイドを参照してください。</p>
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。スケーリングしてはいけません。
リンク/ポート	<p>DBコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> ・ logstash ・ cfssl <p>コンテナは、Dockerネットワーク内で、それにリンクする他のコンテナにポート5432を公開する必要があります。</p> <p>このコンテナは、データベースのレポート用にDockerネットワークの外部にポート55436を公開します。</p>

コンテナ名: blackduck-postgres	
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> logstash: \$HUB_LOGSTASH_HOST cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 該当なし コンテナメモリ: 3 GB コンテナCPU: 1 CPU
ユーザー/グループ	<p>このコンテナはUID 1001として実行されます。コンテナがUID 0(root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 1001:rootに切り替えられます。このコンテナは、他のユーザーIDで開始することはできません。</p>
環境ファイル	該当なし

Documentationコンテナ

コンテナ名: blackduck-documentation	
イメージ名	blackducksoftware/blackduck-documentation:2024.10.0
説明	Documentationコンテナは、アプリケーションのドキュメントを提供します。
スケーラビリティ	このコンテナには1つのインスタンスがあります。スケーリングしてはいけません。
リンク/ポート	<p>このコンテナは、次の他のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> logstash cfssl <p>Documentationコンテナは、それにリンクする他のコンテナにポート8443を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> logstash: \$HUB_LOGSTASH_HOST cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 512 MB コンテナメモリ: 512 MB コンテナCPU: 未指定
ユーザー/グループ	<p>このコンテナはUID 8080として実行されます。コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 8080:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

統合コンテナ

コンテナ名: blackduck-integration	
イメージ名	blackducksoftware/blackduck-integration:2024.10.0
説明	Artifactory Integration と Git SCM プロバイダ統合のスキャン機能を持つ、統合サービス。
スケーラビリティ	<p>このコンテナには複数のインスタンスが存在する場合があります。</p> <ul style="list-style-type: none"> すべてのスキャンサービスが、他のコンテナと同じDockerネットワークにあるという要件 統合サービスは、異なるホストやノードに存在可能
リンク/ポート	<p>このコンテナは、次の他のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> cfssl postgres rabbitmq registration logstash scan <p>統合コンテナは、統合コンテナにリンクする他のコンテナにポート8443を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> postgres: \$HUB_POSTGRES_HOST registration: \$HUB_REGISTRATION_HOST logstash: \$HUB_LOGSTASH_HOST cfssl: \$HUB_CFSSL_HOST rabbitmq: \$RABBIT_MQ_HOST、\$RABBIT_MQ_PORT
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 5,120 MB コンテナメモリ: 5,120 MB コンテナCPU: 1 CPU
ユーザー/グループ	<p>このコンテナはUID 8080として実行されます。コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 8080:rootに切り替えられます。</p> <p>このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Jobrunnerコンテナ

コンテナ名: blackduck-Jobrunner	
イメージ名	blackducksoftware/blackduck-jobrunner:2024.10.0
説明	<p>Job Runnerコンテナは、アプリケーションのすべてのジョブを実行するコンテナです。これには、マッチング、構成表の作成、レポート、データ更新などが含まれます。このコンテナには公開されているポートはありません。</p>

コンテナ名: blackduck-Jobrunner	
スケーラビリティ	このコンテナはスケーリングできます。
リンク/ポート	<p>Job Runnerコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> • postgres • registration • logstash • cfssl
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、個々のサービス名が異なる場合があります。たとえば、別のサービス名を使用して解決された外部PostgreSQLエンドポイントがあるとします。このようなユースケースの場合は、これらの環境変数を設定して、デフォルトのホスト名を上書きすることができます。</p> <ul style="list-style-type: none"> • postgres: \$HUB_POSTGRES_HOST • registration: \$HUB_REGISTRATION_HOST • logstash: \$HUB_LOGSTASH_HOST • cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> • デフォルトの最大Javaヒープサイズ: 4 GB • コンテナメモリ: 4.5 GB • コンテナCPU: 1 CPU
ユーザー/グループ	<p>このコンテナはUID 100として実行されます。コンテナがUID 0 (root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ (GID/fsGroup 0) 内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Logstashコンテナ

コンテナ名: blackduck-logstash	
イメージ名	blackducksoftware/blackduck-logstash:1.0.10
説明	Logstashコンテナは、すべてのコンテナのログを収集して格納します。
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。スケーリングしてはいけません。
リンク/ポート	コンテナは、Dockerネットワーク内で、それにリンクする他のコンテナ/サービスにポート5044を公開する必要があります。
リソース/制約	<ul style="list-style-type: none"> • デフォルトの最大Javaヒープサイズ: 1 GB • コンテナメモリ: 1 GB • コンテナCPU: 未指定
ユーザー/グループ	<p>このコンテナはUID 100として実行されます。コンテナがUID 0 (root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ (GID/fsGroup 0) 内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Matchengineコンテナ

コンテナ名 : matchengine	
イメージ名	blackducksoftware/blackduck-matchengine:2024.10.0
説明	クラウドナレッジベースからコンポーネントマッチ情報を取得します。
スケーラビリティ	このコンテナには複数のインスタンスが存在する場合があります。
リンク/ポート	<ul style="list-style-type: none"> ・ ポートを公開しません。 ・ 外部からクラウドナレッジベースサービスに接続します <p>次のサービスに内部的に接続します。</p> <ul style="list-style-type: none"> ・ cffsl ・ logstash ・ registration ・ postgres ・ rabbitmq
環境変数	<p>環境変数は次のとおりです。</p> <ul style="list-style-type: none"> ・ HUB_CFSSL_POROTHUB_MATCHENGINE_HOST ・ HUB_MAX_MEMORY ・ HUB_REGISTRATION_HOST ・ HUB_REGISTRATION_POROTHUB_POSTGRES_HOST ・ HUB_POSTGRES_PORT ・ RABBIT_MQ_HOST ・ RABBIT_MQ_PORT
リソース/制約	<ul style="list-style-type: none"> ・ デフォルトの最大Javaヒープサイズ: 1 GB ・ コンテナメモリ: 予約値: 1 GB、上限値: 1.5 GB ・ コンテナCPU: 予約値: 0.5、上限値: 1
ユーザー/グループ	<p>このコンテナはUID 100として実行されます</p> <p>コンテナがUID 0 (root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ (GID/fsGroup 0) 内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Rabbitmqコンテナ

コンテナ名 : rabbitmq	
イメージ名	blackducksoftware/rabbitmq:1.2.2
説明	このコンテナにより、バイナリ解析ワーカーに情報を簡単にアップロードできます。また、これによりbomengineコンテナはBOM計算の開始通知を受信できるようになります。Dockerネットワーク内にはポートを公開しますが、Dockerネットワークの外部には公開しません。
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。スケーリングしてはいけません。

コンテナ名 : rabbitmq	
リンク/ポート	<p>このコンテナは、次の他のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> cfssl <p>コンテナは、それにリンクする他のコンテナにポート5671を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> cfssl: \$HUB_CFSSL_HOST
制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 該当なし コンテナメモリ: 1 GB コンテナCPU: 未指定
ユーザー/グループ	<p>このコンテナはUID 100として実行されます。コンテナが UID 0(root)として起動された場合は、メイン プロセスを実行する前に、ユーザーは UID 100:root に切り替えることができます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

Redisコンテナ

コンテナ名 : blackduck-redis	
イメージ名	redis:7.0.9-alpine3.17
説明	<p>このコンテナでは、Black Duckのキャッシュ機能の整合性が向上し、アプリケーションのパフォーマンスが改善します。</p> <p>Redisは、プライマリキャッシュメカニズムとしてデフォルトで有効になっています。</p>
構成	<p>Redisを構成するには、次の設定を使用します。</p> <ul style="list-style-type: none"> Redis設定を構成するためのblackduck-config.env環境ファイル <ul style="list-style-type: none"> Redis設定: <ul style="list-style-type: none"> BLACKDUCK_REDIS_MODE: <p>Redisモードはスタンドアロンまたはセンチネルにすることができます</p> BLACKDUCK_REDIS_TLS_ENABLED: <p>Redisクライアントとサーバーの間でTLS/SSL接続を強制するかどうかを指定できます。</p> Redisスタンドアロンモードにはdocker-compose.ymlを使用します。このモードは、1,024 MBの追加メモリを必要とします Redisセンチネルモードには、docker-compose.ymlとdocker-compose.redis.sentinel.ymlの両方を使用します。このモードは高可用性を提供しますが、3,168 MBの追加メモリを必要とします。 Redisコンテナは、他のサービスと同様にlogstashおよびfilebeatと統合されます。 Redisコンテナには、正常性状態チェックがあります。 Black Duck Hubシステム情報ページには、Redisデバッグ情報が表示されるredis-cacheタブがあります。
スケーラビリティ	コンテナをスケーリングしてはいけません。

コンテナ名: blackduck-redis	
リンク/ポート	Redisコンテナは、次のコンテナ/サービスに接続する必要があります。 <ul style="list-style-type: none"> logstash cfssl
代替ホスト名の環境変数	該当なし
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 該当なし コンテナメモリ: <ul style="list-style-type: none"> スタンドアロン1,024 MB センチネルモード3,168 MB コンテナCPU: 1
ユーザー/グループ	任意のユーザー/グループとして実行できます。次に例を示します。{"runAsUser": 4567, "runAsGroup": 4567}
環境ファイル	blackduck-config.env

Registrationコンテナ

コンテナ名: blackduck-registration	
イメージ名	blackducksoftware/blackduck-registration:2024.10.0
説明	コンテナは、他のコンテナからの登録リクエストを処理する小規模なサービスです。このコンテナはBlack Duck登録サービスへ定期的に接続し、登録の更新を取得します。
スケーラビリティ	コンテナをスケーリングしてはいけません。
リンク/ポート	Registrationコンテナは、次のコンテナ/サービスに接続する必要があります。 <ul style="list-style-type: none"> logstash cfssl <p>コンテナは、それにリンクする他のコンテナにポート8443を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> logstash: \$HUB_LOGSTASH_HOST cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 512 MB コンテナメモリ: 640 MB コンテナCPU: 未指定
ユーザー/グループ	このコンテナはUID 8080として実行されます。コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 8080:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。
環境ファイル	blackduck-config.env

ReversingLabs コンテナ

コンテナ名: rl-service	
イメージ名	blackducksoftware/rl-service:2024.10.0
説明	このコンテナは、マルウェアの有無をバイナリ ファイルで解析します。 現在、このコンテナはBlack Duck — ReversingLabsが有効な場合にのみ使用されます。
スケーラビリティ	このコンテナはスケーリングできます。
リンク/ポート	このコンテナは、次のコンテナ/サービスに接続する必要があります。 <ul style="list-style-type: none"> · cfssl · logstash · rabbitmq · storage · scan · registration
代替ホスト名の環境変数	他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。 <ul style="list-style-type: none"> · cfssl: \$HUB_CFSSL_HOST · logstash: \$HUB_LOGSTASH_HOST · rabbitmq: \$RABBIT_MQ_HOST · storage: \$BLACKDUCK_STORAGE_HOST · scan: \$HUB_SCAN_HOST · registration: \$HUB_REGISTRATION_HOST
リソース/制約	<ul style="list-style-type: none"> · デフォルトの最大Javaヒープサイズ: 該当なし · コンテナメモリ: 6 GB · コンテナCPU: 2 CPU
ユーザー/グループ	このコンテナはUID 1000(rlserviceユーザー名)として実行されます。
環境ファイル	hub-rl.env

Scanコンテナ

コンテナ名: blackduck-scan	
イメージ名	blackducksoftware/blackduck-scan:2024.10.0
説明	scanサービスは、すべてのスキャンデータリクエストの宛先となるコンテナです。
スケーラビリティ	このコンテナはスケーリングできます。
リンク/ポート	このコンテナは、次のコンテナ/サービスに接続する必要があります。 <ul style="list-style-type: none"> · postgres · registration · logstash · cfssl

コンテナ名: blackduck-scan	
	コンテナは、それにリンクする他のコンテナにポート8443を公開する必要があります。
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> • postgres: \$HUB_POSTGRES_HOST • registration: \$HUB_REGISTRATION_HOST • logstash: \$HUB_LOGSTASH_HOST • cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> • デフォルトの最大Javaヒープサイズ: 2 GB • コンテナメモリ: 2.5 GB • コンテナCPU: 1 CPU
ユーザー/グループ	<p>このコンテナはUID 8080として実行されます。コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 8080:rootに切り替えられます。</p> <p>このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	blackduck-config.env

ストレージコンテナ

コンテナ名: blackduck-storage	
イメージ名	blackducksoftware/blackduck-storage:2024.10.0
説明	ファイルのバージョンが複数ある場合、ストレージサービスは、ファイルのアップロード、ファイルのダウンロード、デフォルトファイルの定義を行う機能を多くのユーザーに提供します。
スケーラビリティ	このコンテナはスケーリングできます。
リンク/ポート	<p>このコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> • postgres • registration • rabbitmq • logstash • cfssl
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、個々のサービス名が異なる場合があります。たとえば、別のサービス名を使用して解決された外部PostgreSQLエンドポイントがあるとします。このようなユースケースの場合は、これらの環境変数を設定して、デフォルトのホスト名を上書きすることができます。</p> <ul style="list-style-type: none"> • postgres: \$HUB_POSTGRES_HOST • registration: \$HUB_REGISTRATION_HOST • rabbitmq: \$RABBIT_MQ_HOST • logstash: \$HUB_LOGSTASH_HOST • cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> • デフォルトの最大javaヒープサイズ: 512 MB • コンテナメモリ: 1 GB

コンテナ名	blackduck-storage
ユーザー/グループ	コンテナがUID 0(root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。
環境ファイル	blackduck-config.env

Webappコンテナ

コンテナ名	blackduck-webapp
イメージ名	blackducksoftware/blackduck-webapp:2024.10.0
説明	webappコンテナは、すべてのWeb/UI/APIリクエストの宛先となるコンテナです。また、あらゆるUIリクエストも処理します。図では、webappのポートはDockerネットワークの外部に公開されていません。代わりに、Dockerネットワークの外部に公開されているNGiNXリバースプロキシ(WebServerコンテナを参照)があります。
スケーラビリティ	このコンテナのインスタンスは1つだけである必要があります。スケーリングしてはいけません。
リンク/ポート	<p>webappコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> • postgres • registration • logstash • cfssl <p>コンテナは、それにリンクする他のコンテナにポート8443を公開する必要があります。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> • postgres: \$HUB_POSTGRES_HOST • registration: \$HUB_REGISTRATION_HOST • logstash: \$HUB_LOGSTASH_HOST • cfssl: \$HUB_CFSSL_HOST
リソース/制約	<ul style="list-style-type: none"> • デフォルトの最大Javaヒープサイズ: 2 GB • コンテナメモリ: 2.5 GB • コンテナCPU: 1 CPU
ユーザー/グループ	このコンテナはUID 8080として実行されます。コンテナがUID 0(root)として起動された場合、メインプロセスを実行する前に、ユーザーはUID 8080:rootに切り替えられます。このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。
環境ファイル	blackduck-config.env

Webserverコンテナ

コンテナ名: blackhack-webserver	
イメージ名	blackducksoftware/blackduck-nginx:1.0.32
説明	WebServerコンテナは、アプリケーションを含むコンテナのリバースプロキシです。Dockerネットワークの外部に公開されているポートがあります。これは、HTTPS用に構成されたコンテナです。ここにはHTTPSの構成を可能にする構成ボリュームがあります。 HTTP/2およびTLS 1.3がサポートされています
スケーラビリティ	コンテナをスケーリングしてはいけません。
リンク/ポート	<p>Web Appコンテナは、次のコンテナ/サービスに接続する必要があります。</p> <ul style="list-style-type: none"> webapp cfssl documentation scan authentication <p>このコンテナは、Dockerネットワークの外部にポート443を公開します。</p>
代替ホスト名の環境変数	<p>他のタイプのオーケストレーションで実行する場合、Docker Swarmがデフォルトで使用するのではないホスト名をこれらのコンテナに設定すると便利な場合があります。これらの環境変数は、デフォルトのホスト名を上書きするように設定できます。</p> <ul style="list-style-type: none"> webapp: \$HUB_WEBAPP_HOST cfssl: \$HUB_CFSSL_HOST scan: \$HUB_SCAN_HOST documentation: \$HUB_DOC_HOST authentication: \$HUB_AUTHENTICATION_HOST
リソース/制約	<ul style="list-style-type: none"> デフォルトの最大Javaヒープサイズ: 該当なし コンテナメモリ: 512 MB コンテナCPU: 未指定
ユーザー/グループ	<p>このコンテナはUID 100として実行されます。コンテナがUID 0 (root)として起動された場合は、メインプロセスを実行する前に、ユーザーはUID 100:rootに切り替えられます。</p> <p>このコンテナは、ルートグループ(GID/fsGroup 0)内でも起動される場合、ランダムUIDとして起動することもできます。</p>
環境ファイル	hub-webserver.env、blackduck-config.env