



KubernetesとOpenShiftを使用したBlack Duck SCAのインストール

Black Duck SCA 2025.7.0

Copyright ©2025 by Black Duck.

All rights reserved.このドキュメントの使用はすべて、Black Duck Software, Inc.とライセンシー間のライセンス契約に従うものとします。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。

04-11-2025

目次

まえがき.....	5
Black Duck documentation.....	5
カスタマサポート.....	6
Black Duck コミュニティ.....	6
トレーニング.....	6
Black Duck 包括性と多様性に関する声明.....	7
Black Duck セキュリティへの取り組み.....	7
1. KubernetesとOpenShiftを使用したBlack Duckのインストール.....	8
2. ハードウェア要件.....	9
3. PostgreSQLのバージョン.....	10
一般的な移行プロセス.....	10
4. Helmを使用したBlack Duckのインストール.....	11
Black Duck Helm チャート.....	11
使用する前に.....	11
Black Duck インスタンスの設定.....	12
Black Duck UI の公開.....	13
チャートのインストール.....	14
チャートのアンインストール.....	15
チャートのアップグレード.....	15
設定パラメータ.....	15
共通設定.....	16
認証ポッドの設定.....	18
バイナリ スキャナ ポッドの設定.....	19
BOM エンジン ポッドの設定.....	20
CFSSL ポッドの設定.....	20
Datadog ポッドの設定.....	21
マニュアル ポッドの設定.....	21
統合ポッドの設定.....	22
Job runner ポッドの設定.....	22
Logstash ポッドの設定.....	23
マッチ エンジン ポッドの設定.....	23
PostgreSQL ポッドの設定.....	24
PostgreSQL readiness init コンテナの設定.....	25
PostgreSQL アップグレード ジョブの設定.....	26
RabbitMQ ポッドの設定.....	26
Redis ポッドの設定.....	26
登録ポッドの設定.....	27
スキャン ポッドの設定.....	28
ストレージ ポッドの設定.....	28
Webapp ポッドの設定.....	30

Webserver ポッドの設定.....	31
5. 管理タスク.....	33
Kubernetesでのシークレットの暗号化の構成.....	33
Kubernetesでのシードの生成.....	34
バックアップシードの構成.....	34
Kubernetesでのシークレットローテーションの管理.....	35
Kubernetesシークレットを介して外部データベース認証情報を渡す.....	36
Blackduck Storageのカスタムボリュームの構成.....	36
jobrunnerスレッドプールの設定.....	40
Readiness Probeの設定.....	40
HUB_MAX_MEMORY設定の構成.....	40
Helmを使用したOpenShift上での移行.....	41
JWT 公開/秘密鍵ペアのプロビジョニング.....	41
SCM統合の有効化.....	42
外部データベースにmTLSを構成.....	42
SynopsysctlからHelmチャート導入への移行.....	44

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. KubernetesとOpenShiftを使用したBlack Duckのインストール

KubernetesとOpenShift™は、コンテナを介してクラウド ワークロードを管理するためのオーケストレーション ツールです。Black Duck の展開はHelmチャート経由でサポートされています。次の場所にあるドキュメントとHelmチャートのサンプルを参照してください: [%install dir%/kubernetes/blackduck/](#)

2. ハードウェア要件

サポート対象システム

Black Duck は、インストールと運用に関して以下のシステムをサポートしています。


- ・ 64ビットx86
- ・ ARM64(AArch64)

 注：現在、BDBAとRLサービスでARMシステムはサポート対象外です。

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。

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レプリカ数を調整し、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。

3. PostgreSQLのバージョン


Black Duck 2024.10.0 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 2024.10.0 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 15 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。

Black Duck 2024.10.0 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 2024.10.0 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 15 です。

 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 2025.4.0 では、テスト目的でのみ PostgreSQL 17 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。PostgreSQL 17 のテスト専用サポートは、2025.7.0 ではさらに延長されます。

 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。

4. Helmを使用したBlack Duckのインストール

Helmチャートは、HelmがBlack Duckを導入するのに必要なKubernetesのリソースセットを示しています。Black DuckはHelm 3.5.4をサポートしており、Kubernetesの最小バージョンは1.17です。

Helmチャートはこちらで入手できます:<https://repo.blackduck.com/cloudnative>

Helmを使用してBlack Duckをインストールする手順については、[こちら](#)をクリックしてください。Helmチャートは、Helmパッケージ マネージャを使用して、Kubernetesクラスタ上でBlack Duckの導入をブートストラップします。

Helmを使用したKubernetes上での移行

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、この移行ではCentOS PostgreSQL コンテナの使用がBlack Duck提供のコンテナに置き換えられます。また、blackduck-init コンテナは、blackduck-postgres-waiter コンテナに置き換えられます。

プレーンなKubernetesでは、上書きされない限り、アップグレードジョブのコンテナはルートとして実行されます。ただし、唯一の要件は、ジョブがPostgreSQLデータボリュームの所有者と同じUID（デフォルトではUID=26）で実行されることです。

OpenShiftでは、アップグレードジョブは、PostgreSQLデータボリュームの所有者と同じUIDで実行されることを前提としています。

Black Duck Helm チャート

このチャートは、Helm パッケージ マネージャを使用して、Kubernetes クラスタ上で Black Duck のデプロイをブートストラップします。

 **注：** このドキュメントでは、基本的なデプロイをインストールするクイックスタート プロセスについて説明します。その他の設定オプションについては、Kubernetes のマニュアルを参照してください。

前提条件

- ・ Kubernetes 1.16+
 - ・ storageClass は永続ボリュームを許可するように設定されています。

データの永続性を確保するには、使用中である storageClass の reclaimPolicy を Retain に設定する必要があります。AzureFile(非 CSI バリエーション)は、ファイルとディレクトリのアクセス許可がポッドにマウントされると不変になる SMB マウントとして扱われるため、RabbitMQ のカスタム ストレージ クラスが必要です。
- ・ Helm 3
- ・ レポジトリを、次のローカル Helm レポジトリに追加します。

```
$ helm repo add blackduck https://repo.blackduck.com/cloudnative
```

使用する前に

チャートをローカルに保存

チャートをマシンに保存するには、次のコマンドを実行します。

4. Helmを使用したBlack Duckのインストール・Black Duck インスタンスの設定

```
$ helm pull blackduck/blackduck -d <DESTINATION_FOLDER> --untar
```

これにより、指定されたフォルダ(上記コマンドの `-d` フラグで示されています)にチャートが抽出されます。このフォルダには、アプリケーションのデプロイに必要なファイルが含まれています。

名前空間を作成

以下のコマンドを実行し、名前空間を作成します。この例では `bd` を使用していますが、任意の名前に置き換えることができます。

```
$ BD_NAME="bd"
$ kubectl create ns ${BD_NAME}
```


カスタム TLS シークレットを作成(オプション)

一般的に、Black Duck Helm チャートのインストール前は、カスタム Web サーバー TLS シークレットを指定します。以下のコマンドでシークレットを作成します。

```
$ BD_NAME="bd"
$ kubectl create secret generic ${BD_NAME}-blackduck-webserver-certificate -n ${BD_NAME} --from-file=WEBSERVER_CUSTOM_CERT_FILE=tls.crt --from-file=WEBSERVER_CUSTOM_KEY_FILE=tls.key
```

次に、`values.yaml` の TLS certificate for Black Duck ブロックを更新し、必ず `tlsCertSecretName` の値をコメント解除してください(`tls.crt` ファイルと `tls.key` ファイルが必要です)。`tlsCertSecretName` の値が指定されていない場合は、Black Duck によって独自の証明書が生成されます。

```
tlsCertSecretName: ${BD_NAME}-blackduck-webserver-certificate
```


 注：TLS ターミネーションがアプリケーションから上流で処理されている場合(つまり Ingress リソース経由)、この手順は不要です。

Black Duck インスタンスの設定

適切なデプロイ サイズの選択

Black Duck では、デプロイの適切なサイズ設定に役立つ、1 時間あたりのスキャン数が[Black Duck事前に設定された](#) `yaml` ファイルがいくつか用意されています。これらは、実際の設定を用いた当社のパフォーマンス ラボによってテストされています。ただし、これらは「万能」ではありません。そのため、大量の BDDB スキャン、スニペット スキャン、またはレポートを実行する予定がある場合は、カスタム サイズ設定階層の決定に関するサポートについて CSM に問い合わせてください。

2024.4.x 以降では、GEN04 サイズ設定ファイルを使用する必要があります。

 注：10sph.yaml ファイルは本番環境での使用を意図したものではないため、ローカル テスト以外の目的ではデプロイしないでください。

永続ストレージの設定

Black Duck は、特定のデータをディスクに永続化する必要があります。したがって、インストールでは適切な `storageClass` を利用する必要があります。クラスタにデフォルトの `storageClass` がない、またはオーバーライドする場合は、次のパラメータを更新します。

```
# it will apply to all PVC's storage class but it can be override at container level
storageClass:
```

データの永続性を確保するには、使用中である storageClass の reclaimPolicy を Retain に設定する必要があります。AzureFile(非 CSI バリエーション)は、ファイルとディレクトリのアクセス許可がポッドにマウントされると不変になる SMB マウントとして扱われるため、RabbitMQ のカスタム ストレージ クラスが必要です。


データベースの設定

外部 postgres インスタンス(デフォルト設定)の使用を選択する場合は、values.yaml で次のパラメータを設定する必要があります。

```
postgres.host: ""
postgres.adminUsername: ""
postgres.adminPassword: ""
postgres.userUsername: ""
postgres.userPassword: ""
```

コンテナ化された PostgreSQL インスタンスの利用を選択する場合は、次のパラメータを false に設定します。

```
postgres.isExternal: true
```

 **注：** データベース デプロイの仕様が適切なサイズ階層を満たしていることが重要になります。どのようなデータベース デプロイ方法を選択する場合でも、バックアップを定期的に行い、それらの整合性を定期的に検証するようにしてください。

Black Duck UI の公開

Black Duck ユーザー インターフェイス(UI)には、以下で説明するいくつかの方法でアクセスできます。

NodePort

NodePort は、values.yaml に設定されているデフォルトのサービス型です。カスタム NodePort を使用する場合は、values.yaml 内にある次のパラメータを目的のポートに設定します。

```
# Expose Black Duck's User Interface
exposeui: true
# possible values are NodePort, LoadBalancer or OpenShift (in case of routes)
exposedServiceType: NodePort
# custom port to expose the NodePort service on
exposedNodePort: "<NODE_PORT_TO_BE_USED>"
```

[https://\\$\[NODE_IP\]:\\$\[NODE_PORT\]](https://$[NODE_IP]:$[NODE_PORT]) から Black Duck UI にアクセスできます。

ロードバランサ

value.yaml で exposedServiceType を LoadBalancer に設定すると、Kubernetes は外部ロード バランサ サービスをデプロイするように指示されます。次のコマンドを使用し、Black Duck Web サーバーの外部 IP アドレスを取得できます。

```
$ kubectl get services ${BD_NAME}-blackduck-webserver-exposed -n ${BD_NAME}
```

外部 IP アドレスが保留中として表示される場合は、1 分間待ち、同じコマンドを再入力してください。

[https://\\$\[EXTERNAL_IP\]](https://$[EXTERNAL_IP]) から Black Duck UI にアクセスできます。

Ingress

通常、アプリケーションをユーザーに公開する最も一般的な方法です。まず、Ingress がサービスにルーティングされるため、values.yaml の exposeui を false に設定します。

4. Helmを使用したBlack Duckのインストール・チャートのインストール

```
# Expose Black Duck's User Interface
exposeui: false
```

一般的な Ingress マニフェストは、以下の例のようになります。Ingress コントローラの設定と TLS 証明書自体は、このガイドの範囲外であることに注意してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ${BD_NAME}-blackduck-webserver-exposed
  namespace: ${BD_NAME}
spec:
  rules:
  - host: blackduck.foo.org
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ${BD_NAME}-blackduck-webserver
            port:
              number: 443
  ingressClassName: nginx
```

デプロイ後、UI は Ingress コントローラのパブリック IP にあるポート 443 で利用できるようになります。

OpenShift

values.yaml で exposedServiceType を OpenShift に設定すると、OpenShift はルート サービスをデプロイするように指示されます。

```
# Expose Black Duck's User Interface
exposeui: true
# possible values are NodePort, LoadBalancer or OpenShift (in case of routes)
exposedServiceType: OpenShift
```

次のコマンドを使用し、OpenShift ルートを取得できます。

```
$ oc get routes ${BD_NAME}-blackduck -n ${BD_NAME}
```

[https://\\${ROUTE_HOST}](https://${ROUTE_HOST}) によって Black Duck UI にアクセスできます。

チャートのインストール

Black Duck チャートをインストールするには、次のコマンドを実行します。

```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph" && BD_INSTALL_DIR="<DESTINATION_FOLDER>/blackduck/"
$ helm install ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml
```

i ヒント：helm list を使用してすべてのリリースをリストし、helm get values RELEASE_NAME を使用して指定されたすべての値をリストします。

Helm チャートをインストールする際は、--wait フラグを使用しないでください。--wait フラグはインストールを完了としてマークする前に、すべてのポッドが準備完了になるまで待機します。ただし、インストール後段階に postgres-init ジョブが実行されるまで、ポッドは準備完了になりません。したがって、インストールは決して完了しません。

あるいは、Helm から次のドライラン マニフェストを生成して kubectl apply を使用し、Black Duck をデプロイできます。

```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph" && BD_INSTALL_DIR="<DESTINATION_FOLDER>/blackduck/"
$ helm install ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml --dry-run=client > ${BD_NAME}.yaml

# install the manifest
$ kubectl apply -f ${BD_NAME}.yaml --validate=false
```

チャートのアンインストール

導入ファイルをアンインストール/削除するには:

```
$ helm uninstall ${BD_NAME} --namespace ${BD_NAME}
```

このコマンドは、チャートに関連付けられているすべての Kubernetes コンポーネントを削除し、リリースを削除します。

kubectl を使用し、上記のようにドライランからインストールしていた場合は、次のコマンドでインストールを削除します。

```
$ kubectl delete -f ${BD_NAME}.yaml
```

チャートのアップグレード

新しいバージョンにアップグレードする前に、以下のコマンドを実行し、チャート ミュージアムから最新バージョンのチャートを取得してください。

```
$ helm repo update
$ helm pull blackduck/blackduck -d <DESTINATION_FOLDER> --untar
```

デプロイを更新するには、次の手順に従います。


```
$ BD_NAME="bd" && BD_SIZE="sizes-gen04/120sph"
$ helm upgrade ${BD_NAME} ${BD_INSTALL_DIR} --namespace ${BD_NAME} -f ${BD_INSTALL_DIR}/values.yaml -f ${BD_INSTALL_DIR}/${BD_SIZE}.yaml
```

設定パラメータ

次の表に、Black Duck チャートの設定可能パラメータとそのデフォルト値を示します。

- ・ [共通設定](#)
- ・ [認証ポッド](#)
- ・ [バイナリ スキャナ ポッド](#)
- ・ [BOM エンジン ポッド](#)
- ・ [CFSSL ポッド](#)
- ・ [Datadog ポッド](#)
- ・ [統合ポッド](#)
- ・ [Job runner ポッド](#)
- ・ [Logstash ポッド](#)

- ・ [マッチ エンジン ポッド](#)
- ・ [PostgreSQL ポッド](#)
- ・ [PostgreSQL readiness init コンテナ](#)
- ・ [PostgreSQL アップグレード ジョブ](#)
- ・ [RabbitMQ ポッド](#)
- ・ [Redis ポッド](#)
- ・ [登録ポッド](#)
- ・ [スキャン ポッド](#)
- ・ [ストレージ ポッド](#)
- ・ [Webapp ポッド](#)
- ・ [Webserver ポッド](#)

 注：environs フラグには次のパラメータを設定しないでください。それぞれのフラグを代用してください。

```
Use dataRetentionInDays, enableSourceCodeUpload and maxTotalSourceSizeinMB for the following:
* DATA_RETENTION_IN_DAYS
* ENABLE_SOURCE_UPLOADS
* MAX_TOTAL_SOURCE_SIZE_MB


Use enableAlert, alertName and alertNamespace for the following:
* USE_ALERT
* HUB_ALERT_HOST
* HUB_ALERT_PORT

Use exposedNodePort and exposedServiceType for the following:
* PUBLIC_HUB_WEBSERVER_PORT

Use postgres.isExternal and postgres.ssl for the following:
* HUB_POSTGRES_ENABLE_SSL
* HUB_POSTGRES_ENABLE_SSL_CERT_AUTH

Use enableIPV6 for the following:
* IPV4_ONLY
```

共通設定

パラメータ	説明	デフォルト
registry	イメージ レポジトリ	docker.io/blackducksoftware
imageTag	Black Duck のバージョン	2024.7.1
imagePullSecrets	イメージをプルする際に使用される、1 つ以上のシークレットへの参照	
tlsCertSecretName	証明書を含む Webserver の TLS シークレットの名前(指定されていない場合は証明書が生成されます)	
exposeui	Black Duck Web サーバーのユーザー インターフェイス(UI)を有効化	true
exposedServiceType	Black Duck Web サーバーのサービス型を公開	NodePort

パラメータ	説明	デフォルト
enablePersistentStorage	true の場合、Black Duck には永続ストレージが存在します	true
storageClass	すべての永続ボリューム要求で使用するグローバル ストレージ クラス	
enableLivenessProbe	true の場合、Black Duck には liveness プローブが存在します	true
enableInitContainer	true の場合、Black Duck は必要なデータベースを初期化します	true
enableSourceCodeUpload	true の場合は、環境変数に設定することでソース コードのアップロードが有効になります (environs フラグの値よりも優先されます)。	false
dataRetentionInDays	ソース コードのアップロードのデータ 保持日数	180
maxTotalSourceSizeinMB	ソース コードのアップロードの最大 合計ソース サイズ (MB)	4000
enableBinaryScanner	true の場合は、バイナリ スキャン ワーカーをデプロイすることでバイナリ解析が有効になります	false
enableIntegration	true の場合、環境変数に設定することで Black Duck の統合が有効になります (environs フラグの値よりも優先されます)	false
enableAlert	true の場合、Black Duck Alert サービスは、environ を含む nginx 設定に追加されます。 "HUB_ALERT_HOST:<blackduck_name>-alert.<blackduck_name>.svc	false
enableIPv6	true の場合、環境変数に設定することで IPV6 サポートが有効になります (environs フラグの値よりも優先されます)	true
certAuthCACertSecretName	Black Duck 証明書認証用の独自の 認証局 (CA)	<pre>run this command "kubectl create secret generic -n <namespace> <name>-blackduck- auth-custom-ca --from- file=AUTH_CUSTOM_CA=ca.crt" and provide the secret name</pre>
proxyCertSecretName	Black Duck プロキシ サーバーの 認証局 (CA)	<pre>run this command "kubectl create secret generic -n <namespace> <name>-blackduck-</pre>

4. Helmを使用したBlack Duckのインストール・設定パラメータ

パラメータ	説明	デフォルト
		<pre>proxy-certificate --from-file=HUB_PROXY_CERT_FILE=proxy.crt " and provide the secret name</pre>
proxyPasswordSecretName	Black Duck プロキシ パスワードのシークレット	<pre>run this command "kubectl create secret generic -n <namespace> <name>-blackduck-proxy- password --from- file=HUB_PROXY_PASSWORD_FILE=proxy_password and provide the secret name</pre>
ldapPasswordSecretName	Black Duck LDAP パスワードのシークレット	<pre>run this command "kubectl create secret generic -n <namespace> <name>-blackduck-ldap- password --from- file=LDAP_TRUST_STORE_PASSWORD_FILE=ldap_pa and provide the secret name</pre>
environs	Black Duck 設定に追加する必要がある環境変数	<pre>map e.g. if you want to set PUBLIC_HUB_WEBSERVER_PORT, then it should be --set environs.PUBLIC_HUB_WEBSERVER_PORT=30269</pre>

認証ポッドの設定

パラメータ	説明	デフォルト
authentication.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
authentication.resources.limits.memory	認証コンテナのメモリ制限	1024Mi
authentication.resources.requests.memory	認証コンテナのメモリ リクエスト	1024Mi
authentication.maxRamPercentage	認証コンテナの最大ヒープ サイズ	90
authentication.persistentVolumeClaimName	既存の認証の永続ボリューム要求 (PVC)を指定	
authentication.claimSize	認証の永続ボリューム要求 (PVC) の要求サイズ	2Gi
authentication.storageClass	認証の永続ボリューム要求 (PVC) のストレージ クラス	

パラメータ	説明	デフォルト
authentication.volumeName	既存の認証の永続ボリューム(PV)を指定	
authentication.nodeSelector	ポッド割り当ての認証ノード ラベル	{}
authentication.tolerations	ポッド割り当ての認証ノード容認	[]
authentication.affinity	ポッド割り当ての認証ノード アフィニティ	{}
authentication.podSecurityContext	ポッド レベルの認証セキュリティ コンテキスト	{}
authentication.securityContext	コンテナ レベルの認証セキュリティ コンテキスト	{}

バイナリ スキャナ ポッドの設定

パラメータ	説明	デフォルト
binaryscanner.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	docker.io/sigblackduck
binaryscanner.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	2024.6.3
binaryscanner.resources.limits.Cpu	バイナリ スキャナ コンテナの CPU 制限	1000m
binaryscanner.resources.requests.Cpu	バイナリ スキャナ コンテナの CPU リクエスト	1000m
binaryscanner.resources.limits.memory	バイナリ スキャナ コンテナのメモリ 制限	2048Mi
binaryscanner.resources.requests.memory	バイナリ スキャナ コンテナのメモリ リクエスト	2048Mi
binaryscanner.nodeSelector	ポッド割り当てのバイナリ スキャナ ノード ラベル	{}
binaryscanner.tolerations	ポッド割り当てのバイナリ スキャナ ノード容認	[]
binaryscanner.affinity	ポッド割り当てのバイナリ スキャナ ノード アフィニティ	{}
binaryscanner.podSecurityContext	ポッド レベルのバイナリ スキャナ セキュリティ コンテキスト	{}
binaryscanner.securityContext	コンテナ レベルのバイナリ スキャナ セキュリティ コンテキスト	{}

BOM エンジン ポッドの設定

パラメータ	説明	デフォルト
bomengine.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
bomengine.resources.limits.memory	BOM エンジン コンテナのメモリ制限	1024Mi
bomengine.resources.requests.memory	BOM エンジン コンテナのメモリ リクエスト	1024Mi
bomengine.maxRamPercentage	BOM エンジン コンテナの最大ヒープ サイズ	90
bomengine.nodeSelector	ポッド割り当ての BOM エンジン ノード ラベル	<code>{}</code>
bomengine.tolerations	ポッド割り当ての BOM エンジン ノード 容認	<code>[]</code>
bomengine.affinity	ポッド割り当ての BOM エンジン ノード アフィニティ	<code>{}</code>
bomengine.podSecurityContext	ポッド レベルの BOM エンジン セキュリティ コンテキスト	<code>{}</code>
bomengine.securityContext	コンテナ レベルの BOM エンジン セキュリティ コンテキスト	<code>{}</code>

CFSSL ポッドの設定

パラメータ	説明	デフォルト
cfssl.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
cfssl.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	1.0.28
cfssl.resources.limits.memory	CFSSL コンテナのメモリ制限	640Mi
cfssl.resources.requests.memory	CFSSL コンテナのメモリ リクエスト	640Mi
cfssl.persistentVolumeClaimName	既存の CFSSL の永続ボリューム要求 (PVC) を指定	
cfssl.claimSize	CFSSL の永続ボリューム要求 (PVC) の要求サイズ	2Gi
cfssl.storageClass	CFSSL の永続ボリューム要求 (PVC) のストレージ クラス	
cfssl.volumeName	既存の CFSSL の永続ボリューム (PV) を指定	
cfssl.nodeSelector	ポッド割り当ての CFSSL ノード ラベル	<code>{}</code>
cfssl.tolerations	ポッド割り当ての CFSSL ノード 容認	<code>[]</code>

パラメータ	説明	デフォルト
cfssl.affinity	ポッド割り当ての CFSSL ノード アフィニティ	<code>[]</code>
cfssl.podSecurityContext	ポッド レベルの CFSSL セキュリティ コンテキスト	<code>[]</code>
cfssl.securityContext	コンテナ レベルの CFSSL セキュリティ コンテキスト	<code>[]</code>

Datadog ポッドの設定

パラメータ	説明	デフォルト
datadog.enable	ホストされているお客様の場合のみ true (Values.enableInitContainer は必ず true)	false
datadog.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
datadog.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	1.0.15
datadog.imagePullPolicy	イメージ プル ポリシー	IfNotPresent

マニュアル ポッドの設定

パラメータ	説明	デフォルト
documentation.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
documentation.resources.limits.memory	マニュアル コンテナのメモリ制限	512Mi
documentation.resources.requests.memory	マニュアル コンテナのメモリ リクエスト	512Mi
documentation.maxRamPercentage	マニュアル コンテナのメモリ リクエスト	90
documentation.nodeSelector	ポッド割り当てのマニュアル ノード ラベル	<code>[]</code>
documentation.tolerations	ポッド割り当てのマニュアル ノード 容認	<code>[]</code>
documentation.affinity	ポッド割り当てのマニュアル ノード アフィニティ	<code>[]</code>
documentation.podSecurityContext	ポッド レベルのマニュアル セキュリティ コンテキスト	<code>[]</code>
documentation.securityContext	コンテナ レベルのマニュアル セキュリティ コンテキスト	<code>[]</code>

統合ポッドの設定

パラメータ	説明	デフォルト
integration.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
integration.replicas	統合ポッドのレプリカ数	1
integration.resources.limits.cpu	統合コンテナの CPU 制限	1000m
integration.resources.requests.cpu	統合コンテナの CPU リクエスト	500m
integration.resources.limits.memory	統合コンテナのメモリ制限	5120Mi
integration.resources.requests.memory	統合コンテナのメモリ リクエスト	5120Mi
integration.maxRamPercentage	統合コンテナの最大ヒープ サイズ	90
integration.nodeSelector	ポッド割り当ての統合ノード ラベル	{}
integration.tolerations	ポッド割り当ての統合ノード容認	[]
integration.affinity	ポッド割り当ての統合ノード アフィニティ	{}
integration.podSecurityContext	ポッド レベルの統合セキュリティ コンテキスト	{}
integration.securityContext	コンテナ レベルの統合セキュリティ コンテキスト	{}

Job runner ポッドの設定

パラメータ	説明	デフォルト
jobrunner.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
jobrunner.replicas	Job runner ポッドのレプリカ数	1
jobrunner.resources.limits.cpu	Job runner コンテナの CPU 制限	1000m
jobrunner.resources.requests.cpu	Job runner コンテナの CPU リクエスト	1000m
jobrunner.resources.limits.memory	Job runner コンテナのメモリ制限	4608Mi
jobrunner.resources.requests.memory	Job runner コンテナのメモリ リクエスト	4608Mi
jobrunner.maxRamPercentage	Job runner コンテナの最大ヒープ サイズ	90
jobrunner.nodeSelector	ポッド割り当ての Job runner ノード ラベル	{}
jobrunner.tolerations	ポッド割り当ての Job runner ノード 容認	[]
jobrunner.affinity	ポッド割り当ての Job runner ノード アフィニティ	{}

パラメータ	説明	デフォルト
jobrunner.podSecurityContext	ポッド レベルの Job runner セキュリティ コンテキスト	{}
jobrunner.securityContext	コンテナ レベルの Job runner セキュリティ コンテキスト	{}

Logstash ポッドの設定

パラメータ	説明	デフォルト
logstash.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
logstash.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	1.0.38
logstash.resources.limits.memory	Logstash コンテナのメモリ制限	1024Mi
logstash.resources.requests.memory	Logstash コンテナのメモリ リクエスト	1024Mi
logstash.maxRamPercentage	Logstash の最大ヒープ サイズ	90
logstash.persistentVolumeClaimName	既存の Logstash の永続ボリューム要求 (PVC) を指定	
logstash.claimSize	Logstash の永続ボリューム要求 (PVC) の要求サイズ	20Gi
logstash.storageClass	Logstash の永続ボリューム要求 (PVC) のストレージ クラス	
logstash.volumeName	既存の Logstash の永続ボリューム (PV) を指定	
logstash.nodeSelector	ポッド割り当ての Logstash ノード ラベル	{}
logstash.tolerations	ポッド割り当ての Logstash ノード 容認	[]
logstash.affinity	ポッド割り当ての Logstash ノード アフィニティ	{}
logstash.securityContext	コンテナ レベルの Logstash セキュリティ コンテキスト	{}

マッチ エンジン ポッドの設定

パラメータ	説明	デフォルト
matchengine.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
matchengine.resources.limits.memory	マッチ エンジン コンテナのメモリ制限	4608Mi

4. Helmを使用したBlack Duckのインストール・設定パラメータ

パラメータ	説明	デフォルト
matchengine.resources.requests.memory	マッチ エンジン コンテナのメモリ リクエスト	4608Mi
matchengine.maxRamPercentage	マッチ エンジンの最大ヒープ サイズ	90
matchengine.nodeSelector	ポッド割り当てのマッチ エンジン ノード ラベル	{}
matchengine.tolerations	ポッド割り当てのマッチ エンジン ノード 容認	[]
matchengine.affinity	ポッド割り当てのマッチ エンジン ノード アフィニティ	{}
matchengine.podSecurityContext	ポッド レベルのマッチ エンジン セキュリティ コンテキスト	{}
matchengine.securityContext	コンテナ レベルのマッチ エンジン セキュリティ コンテキスト	{}

PostgreSQL ポッドの設定

パラメータ	説明	デフォルト
postgres.registry	イメージ レポジトリ	docker.io/centos
postgres.isExternal	true の場合は外部 PostgreSQL を使用	true
postgres.host	PostgreSQL ホスト (外部 PostgreSQL を使用する場合にのみ必要)	
postgres.port	PostgreSQL ポート	5432
postgres.pathToPgsqlInitScript	PostgreSQL 初期化スクリプトの完全なファイル パス	external-postgres-init.pgsql
postgres.ssl	PostgreSQL SSL	true
postgres.adminUserName	PostgreSQL 管理者ユーザー名	postgres
postgres.adminPassword	PostgreSQL 管理者ユーザー パスワード	testPassword
postgres.userUserName	PostgreSQL 非管理者ユーザー名	blackduck_user
postgres.userPassword	PostgreSQL 非管理者ユーザー パスワード	testPassword
postgres.resources.requests.cpu	PostgreSQL コンテナの CPU リクエスト (外部 postgres が使用されていない場合)	1000m
postgres.resources.requests.memory	PostgreSQL コンテナのメモリ リクエスト (外部 postgres が使用されていない場合)	3072Mi

パラメータ	説明	デフォルト
postgres.persistentVolumeClaimName	既存の PostgreSQL の永続ボリューム要求(PVC)を指定(外部 postgres が使用されていない場合)	
postgres.claimSize	PostgreSQL の永続ボリューム要求(PVC)の要求サイズ(外部 postgres が使用されていない場合)	150Gi
postgres.storageClass	PostgreSQL の永続ボリューム要求(PVC)のストレージ クラス(外部 postgres が使用されていない場合)	
postgres.volumeName	既存の PostgreSQL の永続ボリューム(PV)を指定(外部 postgres が使用されていない場合)	
postgres.confPersistentVolumeClaimName	既存の PostgreSQL 設定の永続ボリューム要求(PVC)を指定(外部 postgres が使用されていない場合)	
postgres.confClaimSize	PostgreSQL 設定の永続ボリューム要求(PVC)の要求サイズ(外部 postgres が使用されていない場合)	5Mi
postgres.confStorageClass	PostgreSQL 設定の永続ボリューム要求(PVC)のストレージ クラス(外部 postgres が使用されていない場合)	
postgres.confVolumeName	既存の PostgreSQL 設定の永続ボリューム(PV)を指定(外部 postgres が使用されていない場合)	
postgres.nodeSelector	ポッド割り当ての PostgreSQL ノード ラベル	{}
postgres.tolerations	ポッド割り当ての PostgreSQL ノード 容認	[]
postgres.affinity	ポッド割り当ての PostgreSQL ノード アフィニティ	{}
postgres.podSecurityContext	ポッド レベルの PostgreSQL セキュリティ コンテキスト	{}
postgres.securityContext	コンテナ レベルの PostgreSQL セキュリティ コンテキスト	{}

PostgreSQL readiness init コンテナの設定

パラメータ	説明	デフォルト
postgresWaiter.registry	イメージ レポジトリ	
postgresWaiter.podSecurityContext	ポッド レベルの postgres 準備状態 チェック セキュリティ コンテキスト	{}

パラメータ	説明	デフォルト
postgresWaiter.securityContext	コンテナ レベルの postgres 準備状態チェック コンテキスト	<code>{}</code>

PostgreSQL アップグレード ジョブの設定

パラメータ	説明	デフォルト
postgresUpgrader.registry	イメージ レポジトリ	
postgresUpgrader.podSecurityContext	ジョブ レベルの Postgres アップグレーダー セキュリティ コンテキスト	<code>{}</code>
postgresUpgrader.securityContext	コンテナ レベルの Postgres アップグレーダー セキュリティ コンテキスト	<code>{}</code>

RabbitMQ ポッドの設定

パラメータ	説明	デフォルト
rabbitmq.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
rabbitmq.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	1.2.40
rabbitmq.resources.limits.memory	RabbitMQ コンテナのメモリ制限	1024Mi
rabbitmq.resources.requests.memory	RabbitMQ コンテナのメモリ リクエスト	1024Mi
rabbitmq.nodeSelector	ポッド割り当ての RabbitMQ ノード ラベル	<code>{}</code>
rabbitmq.tolerations	ポッド割り当ての RabbitMQ ノード 容認	<code>[]</code>
rabbitmq.affinity	ポッド割り当ての RabbitMQ ノード アフィニティ	<code>{}</code>
rabbitmq.podSecurityContext	ポッド レベルの RabbitMQ セキュリティ コンテキスト	<code>{}</code>
rabbitmq.securityContext	コンテナ レベルの RabbitMQ セキュリティ コンテキスト	<code>{}</code>

Redis ポッドの設定

パラメータ	説明	デフォルト
redis.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
redis.resources.limits.memory	Redis コンテナのメモリ制限	1024Mi
redis.resources.requests.memory	Redis コンテナのメモリ リクエスト	1024Mi

パラメータ	説明	デフォルト
redis.tlsEnalbed	クライアントと Redis 間の TLS 接続を有効化	false
redis.maxTotal	Redis に接続できる同時クライアント接続の最大数	128
redis.maxIdle	追加の接続が解放されずにプール内でアイドル状態を維持できる、同時クライアント接続の最大数	128
redis.nodeSelector	ポッド割り当ての Redis ノード ラベル	{}
redis.tolerations	ポッド割り当ての Redis ノード容認	[]
redis.affinity	ポッド割り当ての Redis ノード アフィニティ	{}
redis.podSecurityContext	ポッド レベルの Redis セキュリティ コンテキスト	{}
redis.securityContext	コンテナ レベルの Redis セキュリティ コンテキスト	{}

登録ポッドの設定

パラメータ	説明	デフォルト
registration.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
registration.requestCpu	登録コンテナの CPU リクエスト	1000m
registration.resources.limits.memory	登録コンテナのメモリ制限	1024Mi
registration.resources.requests.memory	登録コンテナのメモリ リクエスト	1024Mi
registration.maxRamPercentage	登録コンテナの最大ヒープ サイズ	90
registration.persistentVolumeClaimName	既存の登録の永続ボリューム要求 (PVC)を指定	
registration.claimSize	登録の永続ボリューム要求 (PVC) の要求サイズ	2Gi
registration.storageClass	登録の永続ボリューム要求 (PVC) のストレージ クラス	
registration.volumeName	既存の登録の永続ボリューム (PV) を指定	
registration.nodeSelector	ポッド割り当ての登録ノード ラベル	{}
registration.tolerations	ポッド割り当ての登録ノード容認	[]
registration.affinity	ポッド割り当ての登録ノード アフィニティ	{}

パラメータ	説明	デフォルト
registration.podSecurityContext	ポッド レベルの登録セキュリティ コンテキスト	<code>{}</code>
registration.securityContext	コンテナ レベルの登録セキュリティ コンテキスト	<code>{}</code>

スキャン ポッドの設定

パラメータ	説明	デフォルト
scan.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
scan.replicas	スキャン ポッドのレプリカ数	1
scan.resources.limits.memory	スキャン コンテナのメモリ制限	2560Mi
scan.resources.requests.memory	スキャン コンテナのメモリ リクエスト	2560Mi
scan.maxRamPercentage	スキャン コンテナの最大ヒープ サイズ	90
scan.nodeSelector	ポッド割り当てのスキャン ノード ラベル	<code>{}</code>
scan.tolerations	ポッド割り当てのスキャン ノード 容認	<code>[]</code>
scan.affinity	ポッド割り当てのスキャン ノード アフィニティ	<code>{}</code>
scan.podSecurityContext	ポッド レベルのスキャン セキュリティ コンテキスト	<code>{}</code>
scan.securityContext	コンテナ レベルのスキャン セキュリティ コンテキスト	<code>{}</code>

ストレージ ポッドの設定

パラメータ	説明	デフォルト
storage.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
storage.requestCpu	ストレージ コンテナの CPU リクエスト	1000m
storage.resources.limits.memory	ストレージ コンテナのメモリ制限	2048Mi
storage.resources.requests.memory	ストレージ コンテナのメモリ リクエスト	2048Mi
storage.maxRamPercentage	ストレージ コンテナの最大ヒープ サイズ	60
storage.persistentVolumeClaimName	既存のストレージの永続ボリューム 要求 (PVC) を指定	

パラメータ	説明	デフォルト
storage.claimSize	ストレージの永続ボリューム要求 (PVC) の要求サイズ	100Gi
storage.storageClass	ストレージの永続ボリューム要求 (PVC) のストレージ クラス	
storage.volumeName	既存のストレージの永続ボリューム (PV) を指定	
storage.nodeSelector	ポッド割り当てのストレージ ノード ラベル	{}
storage.tolerations	ポッド割り当てのストレージ ノード 容認	[]
storage.affinity	ポッド割り当てのストレージ ノード アフィニティ	{}
storage.podSecurityContext	ポッド レベルのストレージ セキュリティ コンテキスト	{}
storage.securityContext	コンテナ レベルのストレージ セキュリティ コンテキスト	{}
storage.providers	複数のストレージ プラットフォーム をサポートする設定。詳細については、「ストレージ プロバイダ」セクションを参照してください。	[]

ストレージ プロバイダ

ストレージ サービスのプロバイダは、永続性の型とその設定を指します。Black Duck は、ストレージ サービスでツール、アプリケーション レポート、その他の大型 BLOB を管理します。現在、プロバイダの 1 つとしてファイルシステムのみをサポートしています。

```
storage:
  providers:
    - name: <name-for-the-provider> <String>
      enabled: <flag-to-enable/disable-provider> <Boolean>
      index: <index-value-for-the-provider> <Integer>
      type: <storage-type> <String>
      preference: <weightage-for-the-provider> <Integer>
      existingPersistentVolumeClaimName: <existing-persistence-volume-claim-name> <String>
      pvc:
        size: <size-of-the-persistence-disk> <String>
        storageClass: <storage-class-name> <String>
        existingPersistentVolumeName: <existing-persistence-volume-name> <String>
        mountPath: <mount-path-for-the-volume> <String>
```

パラメータ	タイプ	説明	デフォルト
name	String	プロバイダ設定の名前。 例: blackduck-file-storage	
enabled	Boolean	プロバイダ インスタンスの有効/無効化を制御するフラグ	false

4. Helmを使用したBlack Duckのインストール・設定パラメータ

パラメータ	タイプ	説明	デフォルト
index	Integer	プロバイダ設定のインデックス値。例: 1、2、3。	
type	String	ストレージ型。デフォルトは file。	file
preference	Integer	プロバイダ インスタンス設定の重み付けを示す数値。複数のプロバイダ インスタンスが設定されている場合、この値は、どのプロバイダをデフォルトのストレージ オプションとして使用するかを決定するために使用されます。	
existingPersistentVolumeClaimName	String	プロバイダの既存ストレージの永続ボリューム要求を再利用するオプション	
pvc.size	String	永続ボリュームの作成中に使用されるボリュームのサイズ。ストレージ サービスには、最小サイズ 100Gi が推奨されます。	100Gi
pvc.storageClass	String	永続ボリュームに使用されるストレージ クラス	
pvc.existingPersistentVolumeName	String	プロバイダの既存のストレージの永続ボリュームを再利用するオプション	
mountPath	String	プロバイダ ボリュームがマウントされるコンテナ内のパス	
readonly	Boolean	存在する場合は、プロバイダを読み取り専用としてマークできます	false
migrationMode	String	マイグレーションが設定されているかどうかを示します。値は 'NONE'、'DRAIN'、'DELETE' または 'DUPLICATE'	'NONE'

Webapp ポッドの設定

パラメータ	説明	デフォルト
webapp.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
webapp.resources.requests.cpu	Webapp コンテナの CPU リクエスト	1000m

パラメータ	説明	デフォルト
webapp.resources.limits.memory	Webapp コンテナのメモリ制限	2560Mi
webapp.resources.requests.memory	Webapp コンテナのメモリ リクエスト	2560Mi
webapp.maxRamPercentage	Webapp コンテナの最大ヒープ サイズ	90
webapp.persistentVolumeClaimName	既存の Webapp の永続ボリューム要求 (PVC)を指定	
webapp.claimSize	Webapp の永続ボリューム要求 (PVC)の要求サイズ	2Gi
webapp.storageClass	Webapp の永続ボリューム要求 (PVC)のストレージ クラス	
webapp.volumeName	既存の Webapp の永続ボリューム (PV)を指定	
webapp.nodeSelector	ポッド割り当ての Webapp ノード ラベル	{}
webapp.tolerations	ポッド割り当ての Webapp ノード容認	[]
webapp.affinity	ポッド割り当ての Webapp ノード アフィニティ	{}
webapp.podSecurityContext	ポッド レベルの Webapp と Logstash のセキュリティ コンテキスト	{}
webapp.securityContext	コンテナ レベルの Webapp セキュリティ コンテキスト	{}

Webserver ポッドの設定

パラメータ	説明	デフォルト
webserver.registry	コンテナ レベルでオーバーライドされるイメージ レポジトリ	
webserver.imageTag	コンテナ レベルでオーバーライドされるイメージ タグ	2024.7.1
webserver.resources.limits.memory	Webserver コンテナのメモリ制限	512Mi
webserver.resources.requests.memory	Webserver コンテナのメモリ リクエスト	512Mi
webserver.nodeSelector	ポッド割り当ての Webserver ノード ラベル	{}
webserver.tolerations	ポッド割り当ての Webserver ノード 容認	[]
webserver.affinity	ポッド割り当ての Webserver ノード アフィニティ	{}
webserver.podSecurityContext	ポッド レベルの Webserver セキュリティ コンテキスト	{}

4. Helmを使用したBlack Duckのインストール・設定パラメータ

パラメータ	説明	デフォルト
webserver.securityContext	コンテナ レベルの Webserver セキュリティ コンテキスト	

5. 管理タスク

Kubernetesでのシークレットの暗号化の構成

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

 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。

Kubernetesでのシークレットの暗号化の有効化

Kubernetesでシークレットの暗号化を有効にするには、values.yamlオーケストレーションファイルのenableApplicationLevelEncryptionの値をtrueに変更する必要があります。

5. 管理タスク・Kubernetesでのシードの生成

```
# if true, enables application level encryption
enableApplicationLevelEncryption: true
```

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

Kubernetesでのシードの生成

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
```

Kubernetesでのシードの生成

シークレットを作成するKubernetesコマンドラインは多数あります。以下にリストされているコマンドにより、シークレットとその変更の有無をより適切に追跡でき、オンラインシステムでシークレットを操作できることとの互換性が保証されます。シークレットは、Black Duckがアクティブに実行されているKubernetesで作成・削除できます。

```
kubectrl create secret generic crypto-root-seed -n $NAMESPACE --save-config --dry-run=client --
from-file=crypto-root-seed=./root_seed -o yaml | kubectrl apply -f -
```

Kubernetesで前のキーシークレットを削除するには

```
kubectrl delete secret crypto-prev-seed -n $NAMESPACE
```

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ので、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。

Kubernetesでの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、実際にBlack Duckシークレットを暗号/復号化する低レベルのTDEK(テナント復号/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。

Kubernetesでのシークレットローテーション

Kubernetesでは、Black Duckを実行しながら、3つの操作を行うことができます。KubernetesサンプルスクリプトrotateRootSeed.shは、ルートシードをprev_rootに抽出し、新しいルートシードを作成してから、前のシードとルートシードを再作成します。

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。繰り返しになりますが、このクリーンアップは、実行中のKubernetes Black Duckインスタンスに対して実行できます。

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。

Kubernetesシークレットを介して外部データベース認証情報を渡す

外部 PostgreSQL データベースを使用するように Black Duck を設定する際は、values.yaml ファイルにデータベース認証情報を直接保存するのではなく、Kubernetes シークレットを介して渡すことができます。このアプローチでは設定ファイルにプレーンテキスト認証情報を保存することを回避するので、セキュリティが強化されます。

デフォルトの動作を使用(Helmで管理されたシークレット)

デフォルトでは、Helm チャートでは values.yaml ファイル内の adminPassword と userPassword から設定された値を使用し、シークレット <name>-blackduck-db-creds が生成されます。この動作は useHelmChartDbCreds フラグによって制御され、デフォルトで有効になっています。

```
useHelmChartDbCreds: true
```

この方法を引き続き使用する場合、追加手順は不要です。

独自のデータベース認証情報シークレットを入力

自分で認証情報を管理する場合は、values.yaml ファイルの useHelmChartDbCreds を false に設定してください。

```
useHelmChartDbCreds: false
```

その後、Black Duck 導入と同じ名前空間に Kubernetes シークレット <name>-blackduck-db-creds を作成する必要があります。シークレットには、以下のキーを含める必要があります。

- HUB_POSTGRES_ADMIN_PASSWORD_FILE
- HUB_POSTGRES_USER_PASSWORD_FILE

各キーでは、対応するパスワードを含むファイルが指定されている必要があります。以下に例を示します。

```
kubectcl create secret generic -n <namespace> <name>-blackduck-db-creds \
  --from-file=HUB_POSTGRES_ADMIN_PASSWORD_FILE=pg_admin_password_file \
  --from-file=HUB_POSTGRES_USER_PASSWORD_FILE=pg_user_password_file
```

！ 重要： カスタムシークレットが無効または欠落している場合、導入は失敗します。Helm では、values.yaml で指定された認証情報の使用するようにフォールバックされることはありません。

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、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。

複数ボリュームの設定

Kubernetesでカスタムストレージプロバイダを設定するには、以下を含む上書きファイルを作成します。

```
storage:
  providers:
    - name: "file-1"
      enabled: true
      index: 1
      type: "file"
      preference: 20
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "100Gi"
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads"
    - name: "file-2"
      enabled: true
      index: 2
      type: "file"
      preference: 10
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "200Gi"
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads2"
    - name: "file-3"
      enabled: false
      index: 3
      type: "file"
      preference: 30
      readonly: false
      migrationMode: "none"
      existingPersistentVolumeClaimName: ""
      pvc:
        size: "100Gi"
        storageClass: ""
        existingPersistentVolumeName: ""
      mountPath: "/opt/blackduck/hub/uploads3"
```

上記の上書きファイルでは、プロバイダ1と2の両方が有効になっていますが、プロバイダ2の優先度が高くなっているため（優先度の数値が低い）、すべての新しい内容はプロバイダ2に転送されます。

各プロバイダの使用可能な設定は次のとおりです。

設定	詳細
name	デフォルト: なし。 有効な値: すべて。 注記: これは、これらのプロバイダの管理に役立つ表示用ラベルです。
enabled	デフォルト: true プロバイダ1の場合、false (その他の場合)。 有効な値: true または false。

設定	詳細
	注記:プロバイダが有効かどうかを示します。
index	デフォルト:なし。 有効な値:1、2、3。 注記:プロバイダ番号を示します。設定ファイル内の順序は重要ではありません。
type	デフォルト: file。 有効な値:file。 注記:"file"はサポートされる唯一のプロバイダ タイプです。
preference	デフォルト: indexの10倍。 有効な値:0-999。 注記:プロバイダの優先度を設定します。優先度が最大(優先度の数値は最小)のプロバイダには、新しいコンテンツが追加されます。 注:すべてのプロバイダには固有な優先度を指定する必要があります。2つのプロバイダが同じ値にできません。
readonly	デフォルト: false。 有効な値:true または false。 注記: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(優先度の数値が最低)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。 読み取り専用プロバイダでは、データの追加や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。
migrationMode	デフォルト: none。 有効な値:none、drain、delete、duplicate。 注記: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。
existingPersistentVolumeClaimName	デフォルト: ""。 有効な値:任意の有効なk8s識別子。 注記:このボリュームに対し、特定の永続ボリューム要求名を指定できます。
pvc.size	デフォルト: none。 有効な値:任意の有効なサイズ。 注記:ボリュームに対し、利用可能な容量を指定できます。
pvc.storageClass	デフォルト: ""。 有効な値:任意の有効なk8s識別子。 注記:このボリュームに対し、特定のストレージ クラスを指定できます。
pvc.existingPersistentVolumeName	デフォルト: ""。 有効な値:任意の有効なk8s識別子。

設定	詳細
	注記:このボリュームに対し、特定の永続ボリューム名を指定できます。
mountPath	<p>デフォルト: インデックス固有。注記を参照。</p> <p>有効な値:</p> <p>/opt/blackduck/hub/uploads</p> <p>/opt/blackduck/hub/uploads2</p> <p>/opt/blackduck/hub/uploads3</p> <p>注記:</p> <p>特定のプロバイダのためにマウントポイントを設定します。インデックス1のプロバイダには、次のマウント ポイントを指定する必要があります:</p> <p>/opt/blackduck/hub/uploads。インデックス2のプロバイダには、次のマウント ポイントを指定する必要があります:</p> <p>/opt/blackduck/hub/uploads2。インデックス3のプロバイダには、次のマウント ポイントを指定する必要があります:</p> <p>/opt/blackduck/hub/uploads3</p>

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	<p>目的: 移行が進行中でないことを示します。</p> <p>注記: デフォルトの移行モード。</p>
drain	<p>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。</p> <p>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。</p>
delete	<p>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。</p> <p>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。</p>
duplicate	<p>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</p>

移行モード	詳細
	<p>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。</p> <p>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、プルニングジョブで警告が生成されます(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ニングを有効にします。</p> <pre>storage.pruner.orphaned.data.pruning.enable=true</pre>

jobrunnerスレッドプールの設定

Black Duckには、スケジュールされたジョブを実行する定期的プールと、APIやユーザーの操作などの何らかのイベントから開始されるジョブを実行するオンデマンドプールの、2種類のジョブプールがあります。

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)。

Kubernetesでは、次の上書きファイルを使用して、スレッドカウント設定を上書きできます。

```
jobrunner:
  maxPeriodicThreads: 2
  maxPeriodicPrefetch: 1
  maxOndemandThreads: 4
  maxOndemandPrefetch: 2
```

Readiness Probeの設定

values.yamlの以下のブール値フラグを編集することにより、Readiness Probeを有効化あるいは無効化できます：

```
enableLivenessProbe: true
enableReadinessProbe: true
enableStartupProbe: true
```

HUB_MAX_MEMORY設定の構成

Kubernetesベースの展開では、関連するコンテナに対して構成パラメータHUB_MAX_MEMORYが自動で設定されます。この値は、メモリ制限のパーセンテージとして計算され、90%がデフォルト値です。

gen04展開のサイズ設定では、maxRamPercentageが使用される割合を制御します。この設定の値は、HUB_MAX_MEMORYが以前と同じ値になるように選択されています。

Helmを使用したOpenShift上での移行

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、この移行ではCentOS PostgreSQL コンテナの使用がBlack Duck提供のコンテナに置き換えられます。また、blackduck-init コンテナは、blackduck-postgres-waiter コンテナに置き換えられます。

プレーンなKubernetesでは、上書きされない限り、アップグレードジョブのコンテナはルートとして実行されます。ただし、唯一の要件は、ジョブがPostgreSQLデータベースの所有者と同じUID（デフォルトではUID=26）で実行されることです。

OpenShiftでは、アップグレードジョブは、PostgreSQLデータベースの所有者と同じUIDで実行されることを前提としています。

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。

Kubernetes シークレットの作成

1. Kubernetes シークレットを作成します（テンプレート コマンド）。公開/秘密鍵のオプションには、正確なファイルを指定する必要があります。

```
kubectl create secret generic -n <namespace> <name>-blackduck-jwt-keypair --from-file=JWT_PUBLIC_KEY=public_key_file --from-file=JWT_PRIVATE_KEY=private_key_file
```

namespace が bd で name が hub の場合、コマンド例は次のようになります。

```
kubectl create secret generic -n bd hub-blackduck-jwt-keypair --from-file=JWT_PUBLIC_KEY=public-key.pem --from-file=JWT_PRIVATE_KEY=private-key.pem
```

2. namespace で作成された次のシークレットを確認します。

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

namespace が bd で、シークレット名が hub-blackduck-jwt-keypair の場合、想定される出力は次のようになります。

```
kubectl get secrets -n bd
NAME                                TYPE      DATA  AGE
hub-blackduck-jwt-keypair          Opaque    2      7s
```

3. values.yaml の次の行をコメント解除し、シークレット名に応じて name を編集します。

```
jwtKeyPairSecretName: <name>-blackduck-jwt-keypair
```

4. 同じ namespace に Black Duck をデプロイします。以下に例を示します。

```
helm install bd . --namespace bd -f values.yaml -f sizes-gen04/10sph.yaml --set  
exposedNodePort=30000 --set environs.PUBLIC_HUB_WEBSERVER_PORT=30000
```

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。

外部データベースにmTLSを構成

前提条件

Kubernetes導入で外部データベースのmTLSを構成する前に、次のことを確認してください：

1. 環境設定：
 - ・ KubernetesとHelmがインストールされ、適切に構成されている。
 - ・ 永続ボリューム(PV)と永続ボリューム要求(PVC)が環境に設定されている。
2. 導入準備：
 - ・ 使用する正しい導入ファイルを特定してあり、ダウンロードする場所を把握している。
 - ・ 外部データベースを使用するBlack Duck導入の設定に習熟している。
 - ・ 外部データベースでBlack Duck導入を実施するために必要なコマンドが準備できている。
3. mTLS要件：
 - ・ openssl.cnfファイルは、証明書の生成にサーバー上で利用できます。
 - ・ シークレットは、導入手順で提供されるデフォルト値を使用して命名されます。
 - ・ 導入によっては、mTLSにルート証明書、管理者証明書、管理者キー、ユーザー証明書、ユーザーキーの最大5つのシークレットが存在する場合があります。これらを1つ以上使用していない場合は、values.yamlファイル内の対応するエントリを空の文字列(“”)に更新します。
 - ・ 例：ルート証明書を構成するには、value.yamlのpostgres.customCerts.rootCAKeyNameフィールドを更新します。デフォルトでは“HUB_POSTGRES_CA”に設定されています。ルート証明書がない場合は、この値を“”に設定します。
4. 名前空間の一貫性：
 - ・ Black Duck導入とPostgreSQL導入は両方とも同じ名前空間を使用します。

Black Duckデプロイメントへの変更点

次を更新： values.yaml

Black Duck導入用にmTLSを構成するには、必要な設定のシークレットを使用してvalues.yamlファイルを更新する必要があります。次のステップに従います：

1. 次を構成: postgres.sslMode

postgres.sslMode構成オプションをvalues.yamlに追加します。このオプションは、導入に証明書とキーのシークレットを含めるかどうかを判断します。このオプションが構成されていない場合、setenv.shスクリプトは導入から証明書/キーのシークレットを除外します。この値がmTLSを有効にするように設定されていることを確認してください。

2. カスタム証明書のシークレット名を設定

postgres.customCerts.useCustomCertsオプションをvalues.yamlに追加します。これは、外部データベースに接続するために必要な証明書とキーデータ(ルートCA、管理者証明書/キー、ユーザー証明書/キーなど)を含むシークレットの名前を指定します。

3. カスタム証明書データを定義

values.yamlのpostgres.customCertsセクションで、証明書とキーのデータを指定するよう、次の5つのオプションを構成します:

- ・ rootCAKeyName: ルートCA証明書のキー名。
- ・ clientCertName: ユーザー証明書のキー名。
- ・ clientKeyName: ユーザー秘密キーのキー名。
- ・ adminClientCertName: 管理者証明書のキー名。
- ・ adminClientKeyName: 管理者プライベートキーのキー名。

これらの値がシークレット構成内の名前と一致していることを確認してください。

次を更新: postgres-init.yaml

mTLSをサポートするには、postgres-init.yamlファイルに次の変更を加える必要があります:

1. 5つの新しいSSLシークレットのボリュームとボリュームマウントロジックを追加します。
2. PGSSLMODE環境変数を設定するロジックを含めます。
3. PGSSLROOTCERT環境変数を設定するロジックを追加します。
4. 最初にPGSSLCERT環境変数をHUB_POSTGRES_CERTに基づいて、次にHUB_ADMIN_POSTGRES_CERTに基づいて設定します。明確にするため、別のifステートメントを使用する必要があります。
5. 最初にPGSSLKEY環境変数をHUB_POSTGRES_KEYに基づいて、次にHUB_ADMIN_POSTGRES_KEYに基づいて設定します。明確にするため、別のifステートメントを使用する必要があります。
6. /tmp/postgres-init/init.pgsqlを実行するためにロジックを移動し、他のシェルコマンドとグループ化して読みやすさを向上させることで、シェルコマンドロジックを再編成します。

環境を更新

HUB_POSTGRES_ENABLE_SSL_CERT_AUTH: "true"を環境に追加します。

次を更新: postgres.name

データベースポッドを開始後、ポッドのホスト値を取得し、values.yamlのpostgres.hostフィールドを更新します。

postgres.customCerts.useCustomCertsを更新

postgres.customCerts.useCustomCertsをtrueに更新します。

証明書とキーのシークレットを作成

次のコマンドを使用し、証明書とキーのシークレットを作成します：

```
kubectrl create secret generic -n bdbd-blackduck-postgres-certificate --from-  
file=HUB_POSTGRES_CA=root.crt  
--from-file=HUB_POSTGRES_CERT=blackduck_user.crt  
--from-file=HUB_POSTGRES_KEY=blackduck_user.pk8  
--from-file=HUB_ADMIN_POSTGRES_CERT=blackduck_admin.crt  
--from-file=HUB_ADMIN_POSTGRES_KEY=blackduck_admin.pk8
```

SynopsysctlからHelmチャート導入への移行

Black Duckの進化に伴い、Kubernetes導入の管理に向けてsynopsysctlの使用からHelmチャートの使用に移行しています。Helmチャートは、Kubernetes環境におけるBlack Duckの導入、アップグレード、保守に対し、より標準化された柔軟なアプローチを提供します。

既存の導入をアップグレードすると、ボリュームの命名規則とその他の構成の違いによってリスクが生じる可能性があるため、このガイドでは新規インストールをお勧めします。

！ 重要： 移行プロセスの開始前に、データベースとその他の重要なデータを必ずバックアップしてください。このステップは、移行中に予期せぬ問題が発生した場合のデータ損失を防ぐために不可欠です。続行前に、必ずバックアップの整合性を確認してください。


本番環境に進む前に、テスト環境にBlack Duckを導入することを強くお勧めします。これにより、プロセスを検証して潜在的な問題を特定できます。テスト環境は、専用のテストインスタンスにも本番環境からクローンを作成した一時インスタンスにもできます。

内部/外部データベース

外部データベースを使用する場合は、常に1つのBlack Duckインスタンスのみが既存のデータベースに接続することを条件に、既存のデータベースに接続するように新しいインストールを構成できます。または、データベースを新規作成し、既存のデータのバックアップと復元を実行できます。

内部データベースを使用している場合は、現在のデータベースをバックアップして新しいインスタンスに復元する必要があります。

移行プロセス

1. 本番環境からデータベースをバックアップします。
2. 新しい名前空間でHelmを使用し、Black Duckの新規インストールを導入します。
 **注：** 外部データベースを使用する場合は、導入中に既存の外部データベースをポイントするよう、新規インストールを構成します。
3. Black Duckインスタンスが正しく実行されていることを確認します。
4. 本番バックアップを使用し、(内外データベース両方の)データベースを復元します。
5. 徹底的なテストを実行し、機能とデータの整合性を確認してください。
6. ドライランを実行して成功した場合は、本番インスタンスを停止し、新しいインスタンスをポイントするようにDNSルーティングまたはロードバランサーを更新した後、上記のステップを繰り返します。