Kubernetesを使用したBlack Duckのインストール

Black Duck 2023.4.0

目次

まえがき

Black Duck ドキュメント

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。
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環境にBlack Duckソフトウェアをインストールするためのインストール方法は、SynopsysctlとHelmです。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。
- Synopsysctlは、KubernetesおよびRed Hat OpenShiftにBlack Duckソフトウェアを展開するためのクラウドネイティブの管理コマンドラインツールです。

Black Duck 統合に関するドキュメントはConfluenceで入手できます。

カスタマサポート

ソフトウェアまたはドキュメントについて問題がある場合は、Synopsysカスタマサポートに問い合わせてください。

Synopsysサポートには、複数の方法で問い合わせできます。

- オンライン：https://www.synopsys.com/software-integrity/support.html
- 電話：お住まいの地域の電話番号については、サポートページの下段にあるお問い合わせのセクションを参照してください。

サポートケースを開くには、Synopsys Software Integrityコミュニティサイト（https://community.synopsys.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、オンラインカスタマポータルを利用できます。

Synopsys Software Integrityコミュニティ

Synopsys Software Integrityコミュニティは、カスタマサポート、ソリューション、および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Software Integrityグループ(SIG)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SIG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、最新の製品ニュースやSynopsysの最新情報をすべて指先の操作で確認できます。これは、オープンソースの価値を組織内で最大限に高めるように当社の製品やサービスをより上手に活用するのに役立ちます。
- 解決する – SIGの専門家やナレッジベース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 – Software Integrityグループ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@synopsys.comにメールを送信してください。

トレーニング

Synopsys Software Integrity, Customer Education(SIG Edu)は、すべてのBlack Duck教育ニーズに対応する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。

Synopsys Software Integrity, Customer Education(SIG Edu)では、次のことができます。

- 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://community.synopsys.com/s/educationを参照してください。また、Black Duckのヘルプについては、Black Duck UIの[ヘルプ]メニュー()から、[Black Duckチュートリアル]を選択します。

包括性と多様性に関するSynopsysの声明

Synopsysは、すべての従業員、お客様、パートナー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。

1. Synopsysctlを使用したインストール

Kubernetes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。

synopsysctlを使用したインストール

synopsysctlを使用して、Black DuckをKubernetesまたはは OpenShiftにインストールします。

synopsysctlは、KubernetesおよびOpenShiftクラスタでのSynopsysソフトウェアの導入と管理を支援するコマンドラインツールです。synopsysctlをインストールした後には、それを利用してSynopsysソフトウェアを簡単に導入および管理することができます。

- synopsysctlの概要についてはここをクリックしてください。
- synossysctlのインストールと使用方法に関するドキュメントについては、ここをクリックしてください。

注： スケーラビリティのサイジングの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。Synopsysのテクニカルサポート担当者から特に要請がない限り、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。Synopsysの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、Synopsysは可能な範囲で最善のサポートを提供します。

2. ハードウェア要件

以下のパフォーマンスデータは、署名スキャンの永続性が減少したBlack Duck 2022.10.0（デフォルト）とSynopsys Detect 8.0.0を使用して収集されました。SPH値は、署名スキャン、パッケージマネージャdetectorスキャンおよび高速スキャンの組み合わせを使用して計算されています。平均スキャンサイズはお客様によって異なるため、正確なSPHスループットはお客様固有のものです。これらのメトリックは、Google Cloud Platformから収集されました。Google Cloud Platformは、構成ごとに異なるデータベース読み取り/書き込みIOPSを提供します。

10sph	1時間あたりのスキャン回数：50 SPH増加率：400% 1時間あたりのAPI数：2500 プロジェクトバージョン：10000	IOPS： ・ 読み取り：15000 ・ 書き込み：9000 Black Duckサービス： ・ CPU：12コア ・ メモリ：30 GB PostgreSQL： ・ CPU：2コア ・ メモリ：8 GB	合計： ・ CPU：14コア ・ メモリ：38 GB
120sph	1時間あたりのスキャン回数：120 SPH増加率：0% 1時間あたりのAPI数：3000 プロジェクトバージョン：13000	IOPS： ・ 読み取り：15000 ・ 書き込み：15000 Black Duckサービス： ・ CPU：13コア ・ メモリ：46 GB PostgreSQL： ・ CPU：4コア ・ メモリ：16 GB	合計： ・ CPU：17コア ・ メモリ：62 GB
250sph	1時間あたりのスキャン回数：300 SPH増加率：20% 1時間あたりのAPI数：7500 プロジェクトバージョン：15000	IOPS： ・ 読み取り：15000 ・ 書き込み：15000 Black Duckサービス： ・ CPU：17コア ・ メモリ：118 GB PostgreSQL： ・ CPU：6コア ・ メモリ：24 GB	合計： ・ CPU：23コア ・ メモリ：142 GB
500sph	1時間あたりのスキャン回数：650 SPH増加率：30% 1時間あたりのAPI数：18000 プロジェクトバージョン：18000	IOPS： ・ 読み取り：15000 ・ 書き込み：15000 Black Duckサービス：	合計： ・ CPU：38コア ・ メモリ：250 GB

		・ CPU:28コア ・ メモリ:210 GB PostgreSQL: ・ CPU:10コア ・ メモリ:40 GB	
1000sph	1時間あたりのスキャン回数:1400 SPH増加率:40% 1時間あたりのAPI数:26000 プロジェクトバージョン:25000	IOPS: ・ 読み取り:25000 ・ 書き込み:25000 Black Duckサービス: ・ CPU:47コア ・ メモリ:411 GB PostgreSQL: ・ CPU:18コア ・ メモリ:72 GB	合計: ・ CPU:65コア ・ メモリ:483 GB
1500sph	1時間あたりのスキャン回数:1600 SPH増加率:6% 1時間あたりのAPI数:41000 プロジェクトバージョン:28000	IOPS: ・ 読み取り:25000 ・ 書き込み:25000 Black Duckサービス: ・ CPU:60コア ・ メモリ:597 GB PostgreSQL: ・ CPU:26コア ・ メモリ:104 GB	合計: ・ CPU:92コア ・ メモリ:701 GB
2000sph	1時間あたりのスキャン回数:2300 SPH増加率:15% 1時間あたりのAPI数:50000 プロジェクトバージョン:35000	IOPS: ・ 読み取り:60000 ・ 書き込み:25000 Black Duckサービス: ・ CPU:66コア ・ メモリ:597 GB PostgreSQL: ・ CPU:34コア ・ メモリ:136 GB	合計: ・ CPU:100コア ・ メモリ:733 GB

この新しいガイダンスは、現在のBlack Duck 2022.10.0アーキテクチャに基づいています。このガイダンスは、以降のリリースでさらに改良される可能性があります。ご質問やご不明な点がある場合は、製品管理までお問い合わせください。

注： 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。

バイナリスキャン

バイナリスキャンのライセンスがある場合、uploadcacheコンテナ/ポッドのメモリを増やす必要がある場合があります。これは、バイナリスキャナがバイナリを抽出して処理する場所であるためです。デフォルトでは、メモリは512MBに設定されていますが、これは大規模なスキャンには適切ではありません。大規模なバイナリをスキャンする場合は、uploadcacheコンテナ/ポッドのメモリを4 GB以上に増やすことをお勧めします。これを実行するには、YAMLを上書きして、メモリ制限を4096MBに更新します。

Swarmインストールの場合：

```
uploadcache:
  deploy:
    resources:
      limits:
        cpus: ".200"
        memory: "4096M"
      reservations:
        cpus: ".100"
        memory: "4096M"
    replicas: 1
```

Kubernetesインストールの場合：

```
uploadcache:
  replicas: 1
  resources:
    limits:
      cpu: "200m"
      memory: "4096Mi"
    requests:
      cpu: "100m"
      memory: "4096Mi"
```

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 2022.10.0は、新しいPostgreSQLの機能をサポートし、Black Duckサービスのパフォーマンスと信頼性を向上させます。Black Duck 2022.10.0の時点で、内部PostgreSQLコンテナの現在サポートされているPostgreSQLのバージョンは、PostgreSQLコンテナ13です。

以前のバージョンのBlack Duck（2022.10.0より前）からアップグレードする場合は、PostgreSQL 13に移行する必要があります。Black Duck 2022.10.0アップデートは、内部Black Duck PostgreSQLデータベースコンテナをPostgreSQLのバージョン13に移行します。データベースコンテナを使用してOpenShiftに導入する場合は、Black Duckのリリースノートとインストールガイドに記載されているように、1回限りの移行ジョブを実行する必要があります。

注： PostgreSQLのサイジングのガイドラインについては、『Black Duck Hardware Scaling Guidelines』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は、新規インストールにPostgreSQL 14を使用することをお勧めします。PostgreSQL 14.0から14.3には、インデックスが破損するというバグがあるため、サポートされているPostgreSQL 14の最小バージョンは14.4です。

注： Black Duck 2023.4.0では、外部データベースとしてPostgreSQL 15を使用できるように、予備的なサポートが追加されました。このサポートはテスト専用です。本番環境での使用はサポートされていません。

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

このガイダンスは、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Helm3をサポートしており、Kubernetesの最小バージョンは1.13です。

Helmチャートは、https://sig-repo.synopsys.com/artifactory/sig-cloudnativeから入手できます

Helmを使用してBlack Duckをインストールする手順については、ここ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bootstrap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Synopsys提供のコンテナに置き換えられます。また、synopsys-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。

5. Artifactory Integration

概要

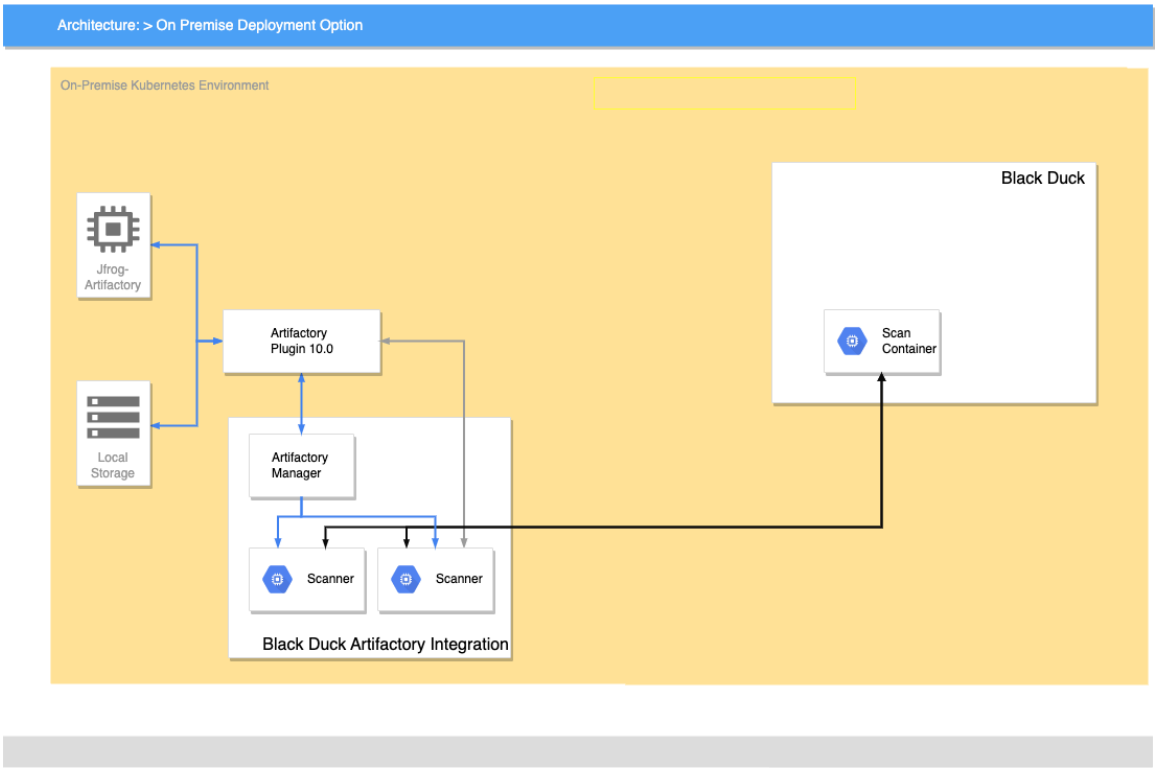
Artifactory Integrationは、ソフトウェアサプライチェーンを保護するための新しいメカニズムです。Artifactoryは通常、そのチェーンの最後のリンクの1つであるため、Artifactoryリポジトリの構成セット内で全アーティファクトを個別にスキャンすることで、個々のサプライチェーンを制御できるようになります。この初期バージョンのArtifactory Integrationでは、スキャンしたArtifactoryリポジトリからのダウンロードにBlack Duckポリシー違反があった場合、そのダウンロードは自動的にブロックされます。高速または両方として定義されているポリシーのみが、Artifactory Integrationに適用されます。

注： Artifactory Integrationを使用すると、ArtifactoryプラグインのScanner機能とInspector機能が無効になります。

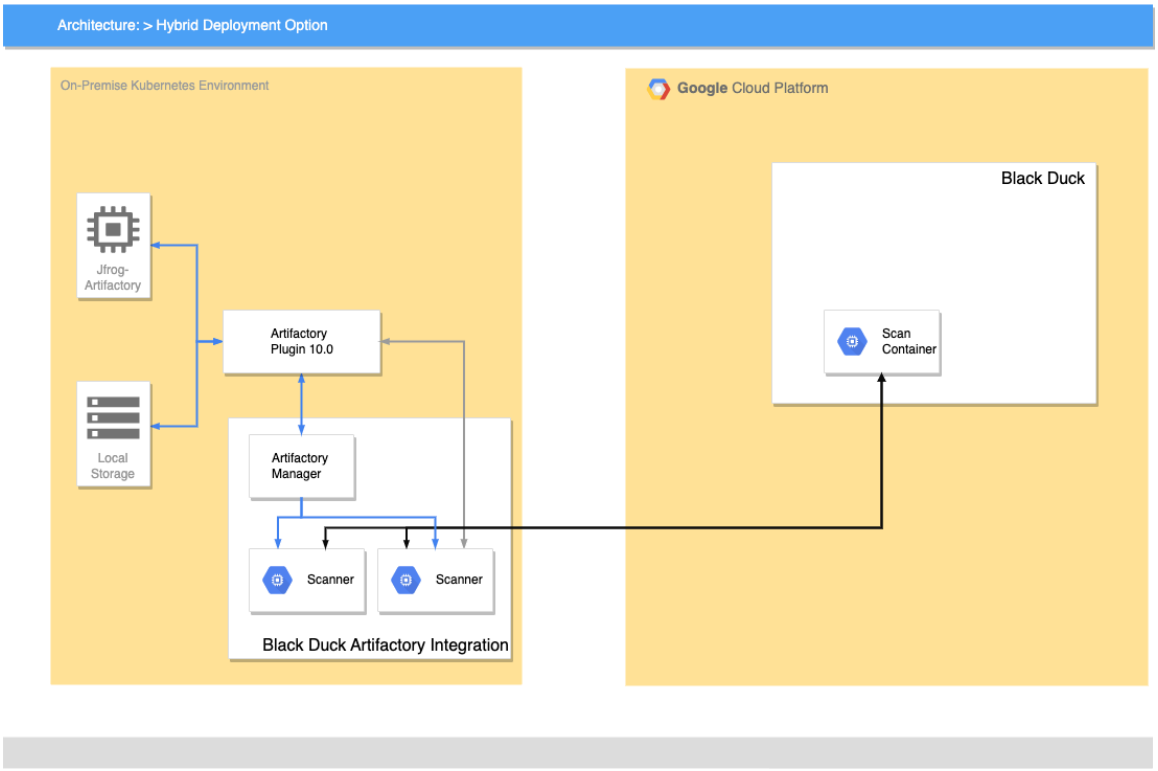
導入の種類

現在、Artifactory Integrationでは、次の2種類の導入がサポートされています。

- 完全なオンプレミス：完全なオンプレミス導入とは、ユーザーのオンプレミス環境で、Black Duckインスタンスとスキャンサービス/Artifactory Integrationの両方を実行することです。



- ハイブリッド：ハイブリッド導入とは、ホスト環境でBlack Duckインスタンスを実行し、ユーザーのオンプレミス環境でスキャンサービス/Artifactory Integrationを実行することです。



Artifactory Integrationの前提条件

カテゴリ	要件
HelmとKubernetes	· Kubernetes 1.9以降 · Helm3 · SynopsysリポジトリをHelmリポジトリに追加します。 `$ helm repo add synopsys https://sig-repo.synopsys.com/artifactory/sig-cloudnative`
クラスタのサイジング	推奨構成は、300 MB未満のアーティファクトの平均サイズに基づきます。最大サイズは2 GB前後です。特定の値が大幅に異なっている場合、それらの数値を再確認する必要があります。 同様に、5スキャナの推奨では、最終的に構成リポジトリのセット全体がスキャンされます。このサイズが5 TBよりも大きく、このバックログのスキャン速度が問題になっている場合は、5スキャナよりも大きな値を設定できます。 スキャナレプリカが5に設定されている場合（推奨）： 表 1 : コア

	必要なコア数	合計コア数
1マネージャ	マネージャあたり2	2

<table>
<tr><th>カテゴリ</th><th>要件</th></tr>
<tr><td></td><td>
<table>
<tr><th></th><th>必要なコア数</th><th>合計
コア数</th></tr>
<tr><td>1メッセージハンドラ
5スキャナとBDBAワーカー</td><td>メッセージハンドラあたり1
スキャナおよびBDBAワーカーあたり1</td><td>1
5</td></tr>
<tr><td></td><td></td><td>8</td></tr>
</table>
表 2 : メモリ
<table>
<tr><th></th><th>必要なメモリ</th><th>合計メモリ</th></tr>
<tr><td>1マネージャ
1メッセージハンドラ
5スキャナ
5 BDBAワーカー</td><td>マネージャあたり4 GB
メッセージハンドラあたり4 GB
スキャナあたり5 GB
BDBAワーカーあたり5 GB</td><td>4 GB
4 GB
25 GB
25 GB</td></tr>
<tr><td></td><td></td><td>58 GB</td></tr>
</table>
ストレージ:
スキャナがスキャン用のアーティファクトをダウンロードできるように、システムに十分な物理ディスク容量があることを確認してください。100 GB以上のディスク容量を推奨します(スキャナレプリカを5に設定した場合)。</td></tr>
<tr><td>ストレージクラス</td><td>Artifactory Integrationの導入には、永続ボリュームをサポートする完全にプロビジョニングされたストレージクラスが必要です。</td></tr>
<tr><td>クラスタのセットアップ</td><td>サービスがアーティファクトをダウンロードし、スキャンしたアーティファクトのプロパティに注釈を付ける場合は、クラスタからArtifactoryサーバーに接続できる必要があります。</td></tr>
<tr><td>永続ボリューム</td><td>スキャナがスキャン用のアーティファクトをダウンロードできるように、Artifactory Integrationの導入では、十分な物理ディスク容量が必要になります。100 GB以上のディスク容量を推奨します(スキャナレプリカを5に設定した場合)。</td></tr>
<tr><td>その他の要件</td><td>Artifactoryプラグイン
・ ターゲットJFrog Artifactory Pro ServerにインストールされたBlackDuck Artifactoryプラグイン。
・ プラグインのインストールは、これまでと同様に現在の手順に従ってください。ScanAsAServiceモジュールに固有のプラグインについては、設定項目に注意してください。</td></tr>
</table>

インストールの順序

ここでは、Artifactory Integrationのインストールについて、順番に手順を概説します。

1. Black Duckインスタンスからアクセストークンを取得し、安全な場所に保存します。
2. Artifactoryインスタンスからアクセストークンを取得し、安全な場所に保存します。

3. Kubernetes環境を準備します。
   a. sig-cloudnativeから最新のArtifactory Integration導入チャートを取得します。
   b. Kubernetesで導入用の名前空間を作成します。
   c. Black DuckおよびArtifactoryのアクセストークンのために、新規作成した名前空間にシークレットを作成します。
4. Artifactory Integrationパラメータを使用してvalues.yamlを編集します。
5. 名前空間にArtifactory Integrationをインストールします。
6. インストールのために、Artifactoryプラグインを準備します。
   a. GitHubからプラグインをダウンロードします。
   b. ダウンロードしたファイルを解凍します。
   c. プラグインファイルをArtifactoryインストールの適切なディレクトリに移動します。
7. ArtifactoryプラグインのblackDuckPlugin.propertiesファイルを必要に応じて編集します。
8. Artifactoryサーバーを再起動します。

Artifactory Integrationのインストール

Helmを介したArtifactory Integrationの導入

1. sig-cloudnativeリポジトリから最新の導入チャートを取得します。この操作により、導入アーカイブ（tar.gz）がダウンロードされます。このアーカイブは、今後の導入手順のために展開する必要があります。

   最新のチャートをダウンロードするには：

   ```
   $ helm repo update
   $ helm pull synopsys/sca-as-a-service
   ```

   チャートを展開してアクセスするには：

   ```
   $ tar xvf sca-as-a-service-x.x.x.tgz
   $ cd sca-as-a-service
   ```

2. 名前空間がまだ作成されていない場合は、名前空間を作成します。

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

3. ArtifactoryインスタンスとBlack Duckインスタンスの両方のために、アクセストークンのシークレットを作成します。

   ```
   $ BD_NAME="bd"
   $ kubectl create secret generic ${BD_NAME}-scaaas-secret-store -n ${BD_NAME} --from-
   literal=scaaas-artifactory-token=<artifactory_token> --from-literal=scaaas-blackduck-
   token=<blackduck_token>
   ```

   Artifactoryアクセストークンスコープ：最小権限（以下にリスト）が満たされている限り、「Admin」ではなく「User」にすることができます。

許可	リポジトリ
・ 読み取り	Artifactory Integrationでスキャンする必要のあるローカルまたはリモートのキャッシュリポジトリ。

・ 注釈	
・ 導入/キャッシュ	スキャンレポートのアップロード先である「ローカル」リポジトリのみ。

4. Artifactory Integrationの導入を設定します。
   a. Artifactory Integrationのhelmチャートをインストールする前にvalues.yamlファイルを更新します。
   b. 次の環境変数に適切な値を設定してください（詳細については、後述の「Artifactory Integrationアプリケーションの設定」セクションを参照してください）。

```
environs:
   BLACKDUCK_SCAAAS_ARTIFACTORY_ANNOTATE_VIOLATING_POLICY_RULES: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_HOST: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_IGNORE_SSL: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_REPOSITORIES: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_DOCKER_REPOSITORIES: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_SCHEME: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_PORT: ""
   BLACKDUCK_SCAAAS_BLACKDUCK_HOST: ""
   BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME: ""
   BLACKDUCK_SCAAAS_BLACKDUCK_TRUST_CERTS:
   BLACKDUCK_SCAAAS_STRUCTURED_LOGGING: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_ENABLED: ""
   BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_REPOSITORY: ""
```

   注： 前述したリストの中で、未設定の環境変数があった場合は、ハッシュタグ（#）を先頭に追加してコメントアウトするか、単に削除することをお勧めします。

   c. 前に作成したシークレットにアクセスできるように、「secrets」セクションが更新されていることを確認します。

```
secrets:
  artifactory:
    token:
      name: bd-scaaas-secret-store
      key: scaaas-artifactory-token
    basicAuth:
      user: {}
      password: {}
    blackduck:
      token:
        name: bd-scaaas-secret-store
        key: scaaas-blackduck-token
```

5. helmチャートを使用してArtifactory Integrationを導入します。

```
$ BD_NAME="bd" && SCAAAS_NAME="scaaas"
$ helm install ${SCAAAS_NAME} sca-as-a-sevice/ --namespace ${BD_NAME}
```

Artifactoryプラグインのインストール

次の手順では、Black Duck Artifactoryプラグインのインストールプロセスと設定プロセスを概説します。

1. blackduck-artifactory-<version>.zipファイルをダウンロードして解凍すると、プラグインプロパティファイルを設定する準備が整います。
2. blackDuckPlugin.propertiesファイルで認証情報に使用するBlack Duck APIトークンを取得します。
3. /plugins/libフォルダにあるblackDuckPlugin.propertiesファイルを使用して、Black Duck認証情報を設定します。

   プラグインの設定についての詳細はこちら

4. pluginsおよびlibフォルダをコピーします。${JFrog_ARTIFACTORY_HOME}/etc/plugins/

   コピーすると次のようになります。

   ${ARTIFACTORY_HOME}/etc/plugins/lib/blackDuckPlugin.properties

5. 次のフォルダのユーザーを変更します。

   - ```
     chown -R 1030:1030 <path-to-blackDuckPlugin.groovy>
     ```

   - ```
     chown -R 1030:1030 <path-to-plugin-libs-directory>
     ```

6. Artifactoryサーバーを再起動します。

接続のテスト

Black Duckプラグインをインストールして設定したら、接続をテストして、プラグインが正しく動作していることを確認することをお勧めします。次のcurlコマンドを使用して、接続をテストします。

```
curl -X GET -u USERNAME:PASSWORD “http://ARTIFACTORY_SERVER/artifactory/api/plugins/execute/
blackDuckTestConfig”
```

DockerとともにインストールされたArtifactory

Docker cpコマンドを実行して、解凍した場所から、プラグインファイルとlibフォルダを移動します ${ARTIFACTORY_HOME}/etc/plugins/.

Artifactoryプラグインの設定

Black Duck Artifactoryプラグインバージョン6.0.0以降では、すべてのプラグインとその設定プロパティがblackDuckPlugin.propertiesファイルに組み込まれています。

プラグインを機能させるには、blackDuckPlugin.propertiesファイルを変更する必要があります。このファイルは、任意のテキストエディタを使用して、プロパティファイルを手動で編集して設定します。デフォルト値を含めて、任意のプロパティを編集できます。したがって、6.0.0より前のプラグインバージョンを使用している場合は、完全な再インストールを強くお勧めします。

注： Black Duck Artifactoryプラグインバージョン6.0.0以降では、Hubを参照するすべてのプロパティが削除され、サポートされなくなりました。

ここでは、blackDuckPlugin.propertiesファイルの重要設定について概要を示します。

Black Duck接続認証情報

プロパティファイルで設定したBlack Duckへの接続が必要です。

少なくとも、プロパティファイルのBlackDuck認証情報の下に、blackduck.api.token=<BD API token>トークンとBlack Duck URLを追加する必要があります。

```
# BlackDuck credentials
blackduck.url=
blackduck.username=
blackduck.password=
blackduck.api.token=
```

アクセストークンを使用しており、Black Duck用のプロキシを使用していない場合、プロパティファイルのCredentialsセクションでは、これが必要とされる唯一の情報になります。

Scan-as-a-Serviceの設定

最も重要なスキャナ設定は、リポジトリリストです。Scan-as-a-Serviceモジュールが確認応答するリポジトリの名前を定義します。これらの設定を調整しないと、定義したポリシーに違反するアイテムをブロックできなくなります。

```
blackduck.artifactory.scaaas.blocking.repos=ext-release-local,libs-release
blackduck.artifactory.scaaas.blocking.docker.repos=ext-docker-repo
```

注： 仮想リポジトリにはコンポーネントが含まれていないため、仮想リポジトリはサポートされていません。仮想リポジトリが参照するローカルリポジトリは、スキャン対象として設定する必要があります。

もう1つの重要な設定は、カットオフ日です。スキャンする必要のない古いアプリケーションがある場合は、この日付によりカットオフを定義します。この場合、この日付よりも古いアプリケーションは無視されます。lastUpdatedの時刻がこの値よりも前になっているアーティファクトは、ブロック戦略の値に関係なく、ブロック戦略の対象とはなりません。

blackduck.artifactory.scan.cutoff.date=2019-05-04T00:00:00.000

プロパティファイル

プラグインを機能させるには、次のプロパティファイルを変更する必要があります。

```
# suppress inspection "UnusedProperty" for whole file

# BlackDuck credentials
blackduck.url=
blackduck.username=
blackduck.password=
blackduck.api.token=
blackduck.timeout=120
blackduck.proxy.host=
blackduck.proxy.port=
blackduck.proxy.username=
blackduck.proxy.password=
blackduck.trust.cert=false

# General
# The date time pattern used by the artifactory to display the scan/inspection timestamp.
# blackduck.artifactory.scan.cutoff.date must comply to this pattern
blackduck.date.time.pattern=yyyy-MM-dd'T'HH:mm:ss.SSS
blackduck.date.time.zone=

# Scanner
blackduck.artifactory.scan.enabled=false
blackduck.artifactory.scan.repos=ext-release-local,libs-release
blackduck.artifactory.scan.repos.csv.path=
blackduck.artifactory.scan.name.patterns=*.jar,*.war,*.zip,*.tar.gz,*.hpi
blackduck.artifactory.scan.binaries.directory.path=
blackduck.artifactory.scan.memory=4096
blackduck.artifactory.scan.dry.run=false
blackduck.artifactory.scan.repo.path.codelocation=true
blackduck.artifactory.scan.repo.path.codelocation.include.hostname=true
blackduck.artifactory.scan.cutoff.date=2020-05-03T00:00:00.000
blackduck.artifactory.scan.cron=0 0/1 * 1/1 * ?

# If metadata.block.repos/metadata.block.repos.csv.path is left blank, all scanned repositories
 are used

blackduck.artifactory.scan.metadata.block=false
blackduck.artifactory.scan.metadata.block.repos=
blackduck.artifactory.scan.metadata.block.repos.csv.path=

# If policy.repos/policy.repos.csv.path is left blank, all scanned repositories are used

blackduck.artifactory.scan.policy.block=true
blackduck.artifactory.scan.policy.repos=
blackduck.artifactory.scan.policy.repos.csv.path=
blackduck.artifactory.scan.policy.severity.types=BLOCKER,CRITICAL,MAJOR,MINOR,TRIVIAL,UNSPECIFIED

# Inspector
```

```
blackduck.artifactory.inspect.enabled=false
blackduck.artifactory.inspect.repos=jcenter-cache
blackduck.artifactory.inspect.repos.csv.path=
blackduck.artifactory.inspect.patterns.bower=*.tar.gz,*.tgz
blackduck.artifactory.inspect.patterns.cocoapods=*.tar.gz
blackduck.artifactory.inspect.patterns.composer=*.zip
blackduck.artifactory.inspect.patterns.conda=*.tar.bz2
blackduck.artifactory.inspect.patterns.cran=*.tar.gz,*.tgz,*.zip
blackduck.artifactory.inspect.patterns.rubygems=*.gem,*.gem.rz,*.gemspec.rz
blackduck.artifactory.inspect.patterns.maven=*.jar
blackduck.artifactory.inspect.patterns.go=*.mod,*.zip
blackduck.artifactory.inspect.patterns.gradle=*.jar
blackduck.artifactory.inspect.patterns.pypi=*.whl,*.tar.gz,*.zip,*.egg
blackduck.artifactory.inspect.patterns.nuget=*.nupkg
blackduck.artifactory.inspect.patterns.npm=*.tgz
blackduck.artifactory.inspect.cron=0 0/1 * 1/1 * ?
blackduck.artifactory.inspect.reinspect.cron=0 0 0 1/1 * ? *
blackduck.artifactory.inspect.retry.count=5
blackduck.artifactory.inspect.metadata.block=false

# If metadata.block.repos/metadata.block.repos.csv.path is left blank, all inspected repositories
 are used
blackduck.artifactory.inspect.metadata.block=false
blackduck.artifactory.inspect.metadata.block.policy.repos=
blackduck.artifactory.inspect.metadata.block.repos.csv.path=

# If policy.repos/policy.repos.csv.path is left blank, all inspected repositories are used
blackduck.artifactory.inspect.policy.block=true
blackduck.artifactory.inspect.policy.repos=
blackduck.artifactory.inspect.policy.repos.csv.path=
blackduck.artifactory.inspect.policy.severity.types=BLOCKER,CRITICAL,MAJOR,MINOR,TRIVIAL,UNSPECIFIED

# Scan-as-a-Service (scaaas)
# If Scan-as-a-Service is enabled, Scanner and Inspector *will* be disabled
blackduck.artifactory.scaaas.enabled=true
blackduck.artifactory.scaaas.blocking.strategy=BLOCK_NONE
blackduck.artifactory.scaaas.blocking.repos=ext-release-local,libs-release
blackduck.artifactory.scaaas.blocking.repos.csv.path=
blackduck.artifactory.scaaas.allowed.patterns=
blackduck.artifactory.scaaas.excluded.patterns=
# Download of items prior to this date will be ALLOWED regardless of the
# value of blackduck.artifactory.scaaas.blocking.strategy
# This date MUST comply with the format in blackduck.date.time.pattern
blackduck.artifactory.scaaas.cutoff.date=
# blocking.docker.repos and blocking.docker.repos.csv.path contain the list of repositories
# that are defined in Artifactory as Docker repositories. These MUST be specified explicitly
# and DO NOT have to be specified as part of blocking.repos or blocking.repos.csv.path.
# If empty, assumes NO repositories are of type Docker.
blackduck.artifactory.scaaas.blocking.docker.repos=ext-docker-repo
blackduck.artifactory.scaaas.blocking.docker.repos.csv.path=

# Analytics
blackduck.artifactory.analytics.enabled=true
```

Artifactory Integrationタスク

Artifactory Integrationのアップグレード

1. 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て、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ます。

```
$ helm repo update
$ helm pull synopsys/sca-as-a-service
```

2. Artifactory Integrationをアップグレードします

```
$ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --namespace ${BD_NAME}
```

Artifactory Integrationの更新

更新は、特定のタイプのアップグレードであり、ENV変数の追加など、変更を同一バージョンに適用できます。更新時には、--reuse-valuesフラグを利用できます。

```
$ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --namespace ${BD_NAME}
```

Artifactory Integrationの再起動

Artifactory Integrationサービスの再起動には、次のオプションを使用できます。

1. ポッドを「0」に縮小してから、目的のレプリカにスケールバックします。

   停止するには：

   ```
   $ kubectl scale deployment <deployment-name> --replicas=0
   ```

   開始するには：

   ```
   $ kubectl scale deployment <deployment-name> --replicas=1
   ```

2. values.yamlでステータスを編集します。

   ステータスを[実行中]から[停止]に変更し、「helm upgrade」を実行します。

   ```
   $ helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --set status="Stopped" --
   namespace ${BD_NAME}
   ```

   ステータスを[停止]から[実行中]に変更し、「helm upgrade」を実行します。

   ```
   helm upgrade ${SCAAAS_NAME} sca-as-a-service/ --reuse-values --set status="Stopped" --
   namespace ${BD_NAME}
   ```

Artifactory Integrationの削除

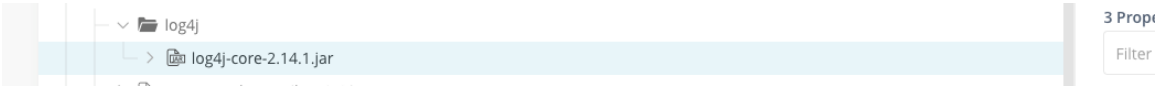
導入ファイルをアンインストール/削除するには：

```
$  helm delete ${SCAAAS_NAME} --namespace ${BD_NAME}
```

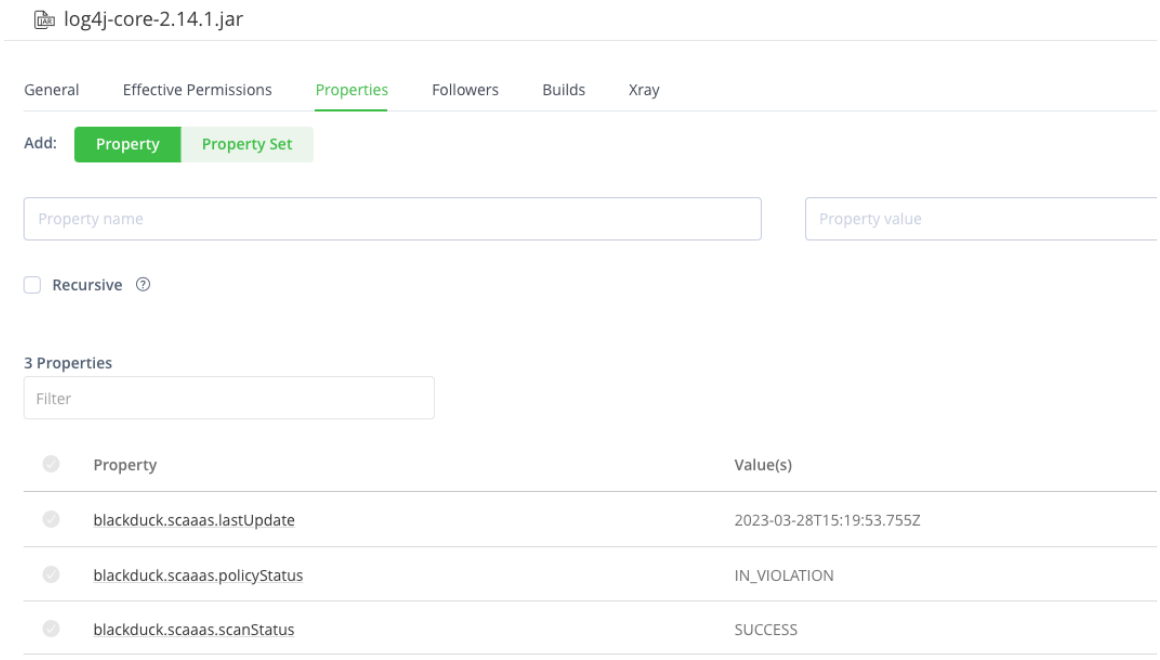
ブロックされたダウンロードの手動上書き

Artifactoryのアイテムが、BlackDuckの定義ポリシーに違反しているため、アイテムを上書きして、アイテムのダウンロードを可能にする場合は、次の手順に従ってください。

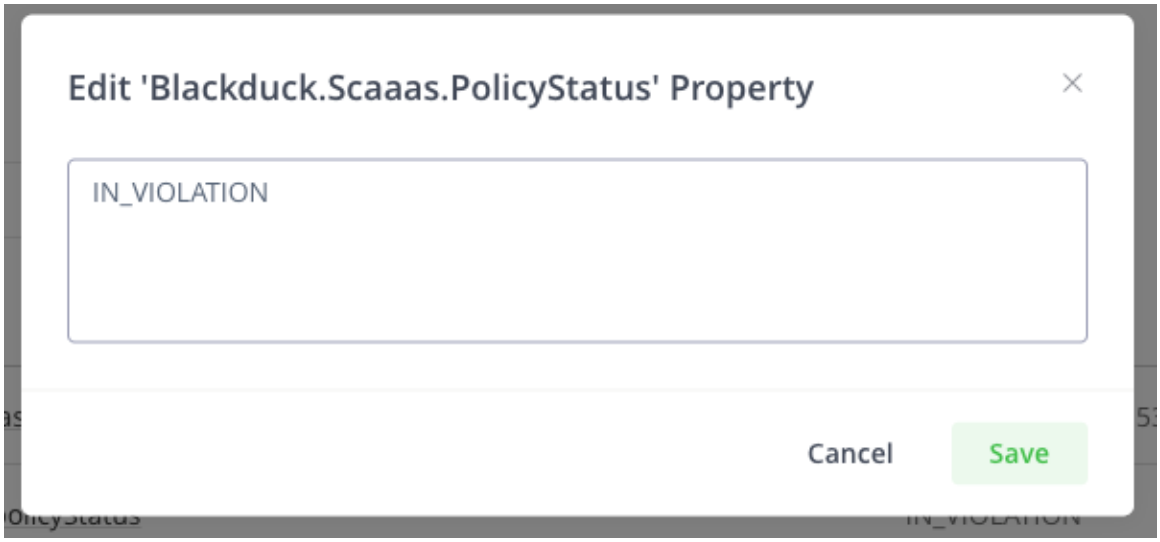
1. Artifactory UIにログインし、上書きする違反アイテムを見つけます。

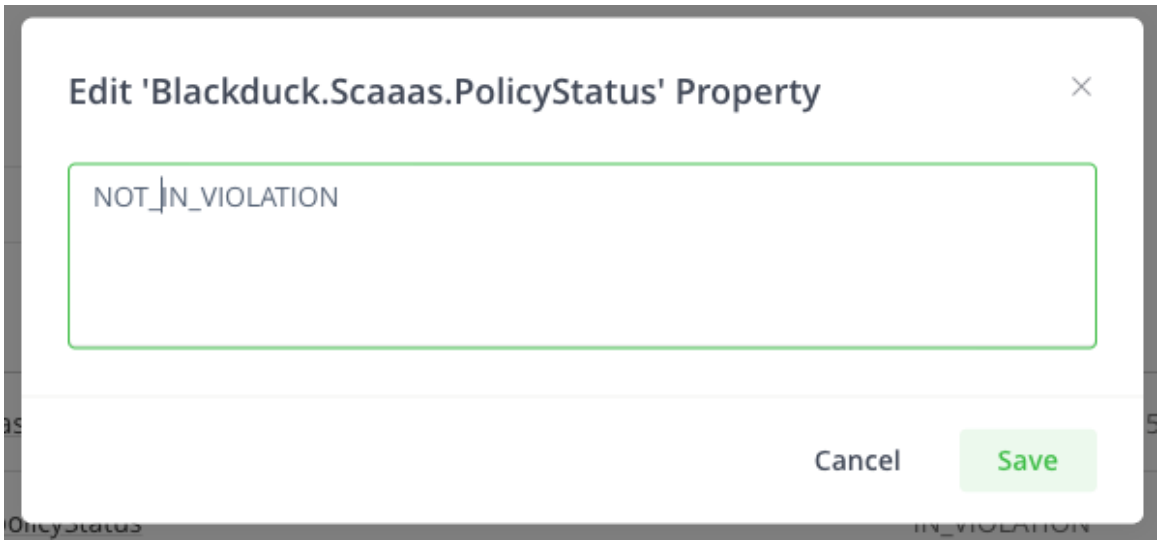


2. 右側のペインから[プロパティ]を選択します。



3. blackduck.scaaas.policyStatusプロパティを選択し、その値をNOT_IN_VIOLATIONに変更して[保存]をクリックします。





これで、Artifactoryプラグインに設定されているブロック戦略に関係なく、アイテムをダウンロードできるようになります。

注： アイテムが更新された場合（新しいバージョンがアップロードされた場合など）は、再スキャンされ、policyStatusがIN_VIOLATIONに再度設定される可能性があります。ファイルを上書きして、ダウンロードできるようにするには、これらの手順を再度実行する必要があります。

バイナリスキャンとコンテナスキャンの無効化

ライセンスでバイナリおよびコンテナのスキャンが許可されていない場合は、values.yamlファイルでBDBAを無効にしてください。この操作を実行した場合に、bdbaworkerコンテナがすでにロードされていた場合、コンテナのロードまたはアンロードは実行されません。署名スキャンのみがサポートされます。

バイナリスキャンとコンテナスキャンを無効にするには、values.yamlファイルの「bdbaworker」セクションを編集し、次のように設定します。

```
enabled: false
```

ファイルを保存し、前述の「Artifactory Integrationの更新」セクションの手順に従って、導入環境に変更を適用します。

Artifactory Integrationの設定

次の表に、Artifactory Integration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。

共有プロパティ

一部のプロパティは、Artifactory IntegrationとArtifactoryプラグインの両方で設定する必要があります。想定どおり確実に動作させるには、これらの値を同一にする必要があります。次の値は、Black DuckとArtifactoryプラグインの両方で同じ値または同等の値にする必要があります。

エクスクルードファイル

- Black Duckの構成: BLACKDUCK_SCAAAS_ARTIFACTORY_EXCLUDE_FILETYPES
- Artifactoryプラグインの設定: blackduck.artifactory.scaaas.excluded.patterns

インクルードファイル

- Black Duckの構成: BLACKDUCK_SCAAAS_ARTIFACTORY_INCLUDE_FILETYPES
- Artifactoryプラグインの設定: blackduck.artifactory.scaaas.allowed.patterns

リポジトリの場所

- Black Duckの構成: BLACKDUCK_SCAAAS_ARTIFACTORY_REPOSITORIES
- Artifactoryプラグインの設定:blackduck.artifactory.scaaas.blocking.reposまたは blackduck.artifactory.scaaas.blocking.repos.csv.path

Dockerリポジトリの場所

- Black Duckの構成: BLACKDUCK_SCAAAS_ARTIFACTORY_DOCKER_REPOSITORIES
- Artifactoryプラグインの設定:blackduck.artifactory.scaaas.blocking.docker.reposまたは blackduck.artifactory.scaaas.blocking.docker.repos.csv.path

カットオフ日

- Black Duckの構成: BLACKDUCK_SCAAAS_SEARCHER_CUTOFF_DATE
- Artifactoryプラグインの設定: blackduck.artifactory.scaaas.cutoff.date

注: Artifactoryプラグインの必須属性。Artifactory Integrationモジュールを正常に初期化します。カットオフ日を設定しない場合(基本的にすべてのアイテムをスキャンし、ブロック戦略を適用する場合)、プラグインのblackDuckPlugin.propertiesファイルで、以下の行をコメントアウトしてください。

```
#blackduck.artifactory.scaaas.cutoff.date=
```

Artifactory Integrationアプリケーションの設定

次の表に、Artifactory Integrationの設定アイテムを示します。

プロパティ名	詳細
BLACKDUCK_RABBIT_SCAOP	説明:Polaris上のSCAのために、使用するVHOSTを有効にします。 デフォルト値: true 必須:いいえ 使用者:

プロパティ名	詳細
	・ scaaas-manager ・ scaaas-scanner ・ rabbitmq
BLACKDUCK_RABBIT_SSL	説明:高速通信にSSLを使用します。 デフォルト値: false 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner ・ rabbitmq
BLACKDUCK_SCAAAS_ARTIFACTORY_ ANNOTATE_VIOLATING_POLICY_RULES	説明:プロパティを有効にすると、そのアーティファクトに対して注釈「violatingPolicyRules」が追加されます。 デフォルト値: false 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ EXCLUDE_FILETYPES	説明:ワイルドカードファイルのカンマ区切り値リスト。この指定により、スキャンするアイテムの検索時に、特定のファイルを除外します。指定しない場合、定義されたリポジトリ内のファイルは除外されなくなります。 デフォルト値: "*sources.jar,*sources.war" 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_HOST	説明:Artifactoryホストの名前。 デフォルト値:該当なし 必須:はい 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ IGNORE_SSL	説明:Artifactory通信では、SSLが無視されます。 デフォルト値: false 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ INCLUDE_FILETYPES	説明:ワイルドカードファイルのカンマ区切り値リスト。この指定により、スキャンするアイテムの検索時に、特定のファイルを含めます。指定しない場合、定義されたリポジトリ内の全ファイルが含まれます。 デフォルト値: "*.hpi,*.jar,*.tar,*.tar.gz,*.war,*.zip" 必須:いいえ 使用者:

プロパティ名	詳細
	• scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_PORT	説明:プロキシArtifactoryホストがリスンするポート。 デフォルト値: -1 必須:いいえ 使用者: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_REPOSITORIES	説明:アーティファクトを検索するリポジトリ。カンマで区切られた文字列(aaa,bbb,ccc...)です。空白の場合、リポジトリはスキャンされません。 デフォルト値:null 必須:いいえ 使用者: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_DOCKER_REPOSITORIES	説明:タグ付き画像を検索するローカルDockerリポジトリ(ArtifactoryをDockerレジストリとして使用する場合)。空白の場合、ローカルDockerリポジトリはスキャンされません。 デフォルト値:null 必須:いいえ 使用者: scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_ENABLED	説明:スキャンレポートの生成とアーティファクトへの保存を有効にします。 デフォルト値: false 必須:いいえ 使用者: • scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_REPOSITORY	説明:スキャンレポートの保存に使用するArtifactory。BLACKDUCK_SCAAAS_ARTIFACTORY_SCAN_REPORT_ENABLEDが有効な場合、設定する必要があります。 デフォルト値:blackduck-scan-results 必須:いいえ 使用者: • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_SCHEME	説明:Artifactoryへの接続に使用するプロトコル。 デフォルト値: HTTPS 必須:はい 使用者: • scaaas-manager • scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_SEARCHER_ADAPTIVE_QUEUE	説明:現在のrabbitMQキューサイズに基づいて、キューに入れるアーティファクトの数を適応的に増減します。 デフォルト値: true

プロパティ名	詳細
	必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SEARCHER_QUEUE_SIZE	説明:スキャンのためにキューに入れるアーティファクトの最大数。 デフォルト値:500 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ SEARCHER_SCHEDULE_DELAY	説明:アーティファクト検索の間隔(秒単位)。最小値30秒。 デフォルト値:60 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_TOKEN	説明:認証用のArtifactory APIトークン(開発導入専用)。 デフォルト値:該当なし 必須:はい 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_ARTIFACTORY_ UPDATED_WINDOW_HOURS	説明:期間を示す時間枠。この期間が経過すると、以前にスキャンしたが、スキャン日以降に再びアップロードされたアーティファクトが検索されます。Artifactory Integrationがダウンしない非常に安定した環境では、この値は2時間(最小値)のような小さな値になる可能性があります。 デフォルト値:48時間 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_ARTIFACTORY_ URI_PATH	説明:Artifactoryでアーティファクトを検索するパス。これは、全API呼び出しのArtifactoryホスト名の末尾に追加されます。デフォルト値は「artifactory」です。 デフォルト値:該当なし 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_HOST	説明:Blackduckインスタンスの名前。 デフォルト値:該当なし 必須:はい 使用者: ・ scaaas-scanner

プロパティ名	詳細
BLACKDUCK_SCAAAS_BLACKDUCK_PORT	説明:Black Duckインスタンスが実行されているポート。 デフォルト値:該当なし 必須:はい 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME	説明:Blackduckインスタンスへの接続に使用するプロトコル。 デフォルト値:該当なし 必須:はい 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_TOKEN	説明:スキャン認証用のBlackduck APIトークン(開発導入専用)。 デフォルト値:該当なし 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_BLACKDUCK_ TRUST_CERTS	説明:Black Duckインスタンスから取得した証明書を自動的に信頼します デフォルト値:false 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_COMPRESS_ALL_ MQ_MESSAGES	説明:有効にすると、すべてのrabbitMQメッセージが圧縮されます。 デフォルト値: false 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_DETECT_ BDBA_TIMEOUT	説明:Detect BDBAスキャンタイムアウト(秒単位)。 デフォルト値:3600 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_DETECT_TIMEOUT	説明:Detectスキャンタイムアウト(秒単位)。 デフォルト値:600 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_DOWNLOAD_ TIMEOUT_MINUTES	説明:スキャナのダウンロードが例外で終了し、このタイムアウト値よりも時間がかかった場合、そのアイテムはFAILEDとしてマークされます。

プロパティ名	詳細
	デフォルト値:30分 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_FAILED_COUNT	説明:アーティファクトの最大再試行回数。 デフォルト値:3 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_FAILED_ TIMEOUT_HOURS	説明:スキャンを失敗と宣言して、再びキューに入れるまで待機する時間数。指定できる値は、6～96(これらの値を含む)です。 デフォルト値:48 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_MANAGER_SEARCHER_QUEUE_THRESHOLD_HIGH	説明:キュー内のメッセージに基づく最大しきい値のパーセンテージ。キューがいっぱい(またはクローズ)の場合は、キューサイズを小さくします。 デフォルト値:80 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_MANAGER_SEARCHER_QUEUE_THRESHOLD_LOW	説明:キュー内のメッセージに基づく最小しきい値のパーセンテージ。キューが空(またはクローズ)の場合は、キューサイズを大きくします。 デフォルト値:20 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_PROCESSING_TIMEOUT_HOURS	説明:スキャンを処理中と宣言して、スキャンのために再びキューに入れるまで待機する時間数。 デフォルト値:2 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_REPOSITORY_TYPE	説明:スキャンするリポジトリのタイプ。 デフォルト値: ARTIFACTORY 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner
BLACKDUCK_SCAAAS_REQUEST_PREFETCH	説明:RabbitMQコンシューマプリフェッチ値。コンシューマ1人あたりの処理メッセージ量を指定します。

プロパティ名	詳細
	デフォルト値:1 必須:いいえ 使用者: ・ scaaas-scanner
BLACKDUCK_SCAAAS_SEARCHER_ CUTOFF_DATE	説明:設定した場合、設定したカットオフ日の後に作成されたアーティファクトのみがスキャン対象となります。指定形式は、YYYY-MM-DDTHH:MMです。 デフォルト値:該当なし 必須:いいえ 使用者: ・ scaaas-manager
BLACKDUCK_SCAAAS_STALL_ON_FAILURE	説明:Javaをバックグラウンドで実行し、それを監視することで、Javaの起動に問題がある場合、クラッシュが発生した場合に、オプションで停止できるようになっています。このオプションを有効にしても、デバッグと分析のためにコンテナに接続することは可能です。 デフォルト値: false 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner
DETECT_LATEST_RELEASE_VERSION	説明:スキャンに使用するDetectのバージョン。Artifactory IntegrationではDetect 8.2.0以上が必要です。 デフォルト値: <value as configured from synopsys-detect dependency> 必須:いいえ 使用者: ・ scaaas-scanner
RABBIT_MQ_PORT	説明:rabbitMQ通信に使用するポート。 デフォルト値:5672 必須:はい 使用者: ・ scaaas-manager ・ scaaas-scanner ・ rabbitmq
RABBITMQ_DEFAULT_VHOST	説明:rabbitMQ通信に使用する仮想ホスト。 デフォルト値: blackduck 必須:いいえ 使用者: ・ scaaas-manager ・ scaaas-scanner ・ rabbitmq

Artifactoryプラグインの設定

次の表では、Artifactoryプラグインで、Artifactory Integrationのために使用できる設定項目を説明します。

パラメータ	詳細
blackduck.artifactory.scaaas.enabled	ScanAsAServiceモジュールが有効かどうかを示します。これをtrueに設定すると、ScanおよびInspectモジュールは、それぞれのenabled値に関係なく強制的に無効にされます。 デフォルト値： false
blackduck.artifactory.scaaas. blocking.strategy	ダウンロード要求のためにプラグインが使用するブロック戦略。次の値がサポートされています。 BLOCK_OFF: ダウンロード要求にはブロックは適用されません。アクションがログに記録されます。これは、プラグインの実行に問題がある場合は、「ドライラン」モードまたはフォールバックモードとして使用できます。 BLOCK_NONE: まだスキャンされていないアーティファクトには、ブロックは適用されません。アーティファクトがスキャンされ、blackduck.scaaas.policyStatusの注釈にIN_VIOLATION値が付いている場合にのみ、ブロックが適用されます。 BLOCK_ALL: スキャンされたアーティファクトで、なおかつ、blackduck.scaaas.policyStatusの注釈にIN_VIOLATION値が付いているアーティファクトにブロックが適用されます。また、まだスキャンされていないアーティファクトにもブロックが適用されます。これは最も制限が厳しいモードです。 デフォルト値： BLOCK_NONE
blackduck.artifactory.scaaas.blocking.repos	リポジトリ名のカンマ区切り値リスト。ここに指定したリポジトリが、blackduck.artifactory.scaaas.blocking.strategyに設定したブロックの対象になります。これは有効なリポジトリのリストに変更する必要があります。変更しないと、ブロックが正しく適用されない可能性があります。この値が設定されている場合は、blackduck.artifactory.scaaas.blocking.repo.csv.pathを空白のままにする必要があります。 デフォルト値：ext-release-local、libs-release
blackduck.artifactory.scaaas. blocking.repos.csv.path	リポジトリ名のカンマ区切り値リストが保存されたファイルへのパス。リストに指定したリポジトリが、blackduck.artifactory.scaaas.blocking.strategyに設定したブロックの対象になります。この値が設定されている場合は、blackduck.artifactory.scaaas.blocking.repoを空白のままにする必要があります。
blackduck.artifactory.scaaas. blocking.docker.repos	Dockerリポジトリ名のカンマ区切り値リスト。ここに指定したリポジトリが、blackduck.artifactory.scaaas.blocking.strategyに設定したブロックの対象になります。これはDockerリポジトリ値のリストに変更する必要があります。変更しないと、ブロックが正しく適用されない可能性があります。この値が設定されている場合、blackduck.artifactory.scaaas.blocking.docker.repos.csv.pathを空白のままにする必要があります。 デフォルト値：ext-docker-repo
blackduck.artifactory.scaaas. blocking.docker.repos.csv.path	Dockerリポジトリ名のカンマ区切り値リストが保存されたファイルへのパス。リストに指定したリポジトリが、blackduck.artifactory.scaaas.blocking.strategyに設定したブロックの対象になります。この値が設定されている場

パラメータ	詳細
	合、blackduck.artifactory.scaaas.blocking.docker.reposを空白のままにする必要があります。
blackduck.artifactory.scaaas. allowed.patterns	ファイルパターンのワイルドカードフィルタ。条件に適合したアーティファクトは、設定したブロック戦略の適用対象になります。空の値は、全ファイルが含まれることを示しています。これはカンマ区切りの値リストとして提供します。
blackduck.artifactory.scaaas. excluded.patterns	ファイルパターンのワイルドカードフィルタ。条件に適合したアーティファクトは、設定したブロック戦略の適用対象から除外されます。空の値は、除外されるファイルがないことを示しています。これはカンマ区切りの値リストとして提供します。
blackduck.artifactory.scaaas.cutoff.date	lastUpdatedの時刻がこの値よりも前になっているアーティファクトは、ブロック戦略の値に関係なく、ブロック戦略の対象とはなりません。

6. 管理タスク

Kubernetesでのシークレットの暗号化の構成

Black Duckは、システム内で重要なデータの、保存データ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。

```
# 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で作成および削除できます。

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

Kubernetesで前のキーシークレットを削除するには

```
kubectl 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。

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、Black Duckのシークレットを実際に暗号化および復号化する、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。

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。

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（優先度の数値は小さい）、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：プロバイダ1の場合はtrue、その他の場合はfalse。 有効な値：trueまたはfalse。 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なることは許されません。
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。 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。

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

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。

移行モード	詳細
	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、プルー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ーニングを有効にします。 `storage.pruner.orphaned.data.pruning.enable=true`

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）。

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
```