



# 使用 Kubernetes 安装 Black Duck

Black Duck 2023.10.0

Synopsys 版权所有 ©2023。

保留所有权利。本文档的所有使用均受 Black Duck Software, Inc. 和被许可人之间的许可协议约束。未经 Black Duck Software, Inc. 事先书面许可，不得以任何形式或任何方式复制或传播本文档的任何内容。

Black Duck、Know Your Code 和 Black Duck 徽标是 Black Duck Software, Inc. 在美国和其他司法管辖区的注册商标。Black Duck Code Center、Black Duck Code Sight、Black Duck Hub、Black Duck Protex 和 Black Duck Suite 是 Black Duck Software, Inc. 的商标。所有其他商标或注册商标均为其各自所有者的独占财产。

25-01-2024

# 内容

前言.....	4
Black Duck 文档.....	4
客户支持.....	4
Synopsys Software Integrity 社区.....	5
培训.....	5
Synopsys 关于包容性和多样性的声明.....	5
Synopsys 安全注释.....	6
1. 使用 Kubernetes 安装 Black Duck.....	7
2. 硬件要求.....	8
3. PostgreSQL 版本.....	10
4. 使用 Helm 安装 Black Duck.....	11
5. Artifactory 集成.....	12
Artifactory 集成前提条件.....	12
安装顺序.....	13
安装 Artifactory 集成.....	14
安装 Artifactory 插件（已弃用）.....	15
安装 Artifactory 集成插件.....	16
配置 Artifactory 插件（已弃用）.....	17
配置 Artifactory 集成插件.....	19
测试连接.....	19
Artifactory 集成任务.....	20
配置 Artifactory 集成.....	23
6. 管理任务.....	25
在 Kubernetes 中配置密钥加密.....	25
在 Kubernetes 中生成种子.....	26
配置备份种子.....	26
在 Kubernetes 中管理密钥轮换.....	27
为 Blackduck 存储配置自定义卷.....	27
配置 jobrunner 线程池.....	30

# 前言

## Black Duck 文档

Black Duck 的文档包括在线帮助和以下文档：

标题	文件	说明
发行说明	release_notes.pdf	包含与当前版本和先前版本中的新功能和改进功能、已解决问题和已知问题有关的信息。
使用 Docker Swarm 安装 Black Duck	install_swarm.pdf	包含有关使用 Docker Swarm 安装和升级 Black Duck 的信息。
使用 Kubernetes 安装 Black Duck	install_kubernetes.pdf	包含有关使用 Kubernetes 安装和升级 Black Duck 的信息。
使用 OpenShift 安装 Black Duck	install_openshift.pdf	包含有关使用 OpenShift 安装和升级 Black Duck 的信息。
入门	getting_started.pdf	为初次使用的用户提供了有关使用 Black Duck 的信息。
扫描最佳做法	scanning_best_practices.pdf	提供扫描的最佳做法。
SDK 入门	getting_started_sdk.pdf	包含概述信息和样本使用案例。
报告数据库	report_db.pdf	包含有关使用报告数据库的信息。
用户指南	user_guide.pdf	包含有关使用 Black Duck 的 UI 的信息。

在 Kubernetes 或 OpenShift 环境中安装 Black Duck 软件的安装方法是 Helm。单击以下链接查看文档。

- [Helm](#) 是 Kubernetes 的软件包管理器，可用于安装 Black Duck。Black Duck 支持 Helm3，Kubernetes 的最低版本为 1.13。

Black Duck 集成文档位置：

- <https://sig-product-docs.synopsys.com/bundle/integrations-detect/page/integrations/integrations.html>
- [https://sig-product-docs.synopsys.com/category/cicd\\_integrations](https://sig-product-docs.synopsys.com/category/cicd_integrations)

## 客户支持

如果您在软件或文档方面遇到任何问题，请联系 Synopsys 客户支持。

您可以通过以下几种方式联系 Synopsys 支持：

- 在线：<https://www.synopsys.com/software-integrity/support.html>
- 电话：请参阅我们的[支持页面](#)底部的“联系我们”部分以查找您当地的电话号码。

要打开支持案例，请登录 Synopsys Software Integrity 社区网站，网址为：<https://community.synopsys.com/s/contactsupport>。

另一个可随时使用的方便资源是[在线客户门户](#)。

## Synopsys Software Integrity 社区

Synopsys Software Integrity 社区是我们提供客户支持、解决方案和信息的主要在线资源。该社区允许用户快速轻松地打开支持案例，监控进度，了解重要产品信息，搜索知识库，以及从其他 Software Integrity Group (SIG) 客户那里获得见解。社区中包含的许多功能侧重于以下协作操作：

- 连接 - 打开支持案例并监控其进度，以及监控需要工程或产品管理部门协助的问题
- 学习 - 其他 SIG 产品用户的见解和最佳做法，使您能够从各种行业领先的公司那里汲取宝贵的经验教训。此外，客户中心还允许您轻松访问 Synopsys 的所有最新产品新闻和动态，帮助您更好地利用我们的产品和服务，最大限度地提高开源组件在您的组织中的价值。
- 解决方案 - 通过访问 SIG 专家和我们的知识库提供的丰富内容和产品知识，快速轻松地获得您正在寻求的答案。
- 分享 - 与 Software Integrity Group 员工和其他客户协作并进行沟通，以众包解决方案，并分享您对产品方向的想法。

[访问客户成功社区](#)。如果您没有帐户或在访问系统时遇到问题，请单击[此处](#)开始，或发送电子邮件至 [community.manager@synopsys.com](mailto:community.manager@synopsys.com)。

## 培训

Synopsys Software Integrity 的客户教育 (SIG Edu) 板块是满足您的所有 Black Duck 教育需求的一站式资源。它使您可以全天候访问在线培训课程和操作方法视频。

每月都会添加新视频和课程。

在 Synopsys Software Integrity 的客户教育 (SIG Edu) 板块，您可以：

- 按照自己的节奏学习。
- 按照您希望的频率回顾课程。
- 进行评估以测试您的技能。
- 打印完成证书以展示您的成就。

要了解更多信息，请访问 <https://community.synopsys.com/s/education>，或者，要获取 Black Duck 的帮助

信息，请选择 Black Duck 教程（位于 Black Duck UI 的“帮助”菜单（）中）。

## Synopsys 关于包容性和多样性的声明

Synopsys 致力于打造一个包容性的环境，让每位员工、客户和合作伙伴都感到宾至如归。我们正在审查并移除产品中的排他性语言以及支持面向客户的宣传材料。我们的举措还包括通过内部计划从我们的工程和工作环境中移除偏见语言（包括嵌入我们软件和 IP 中的术语）。同时，我们正在努力确保我们的 Web 内容和软件应用程序可供不同能力的人使用。由于我们的 IP 实施了行业标准规范，目前正在审查这些规范以移除排他性语言，因此您可能仍在我们的软件或文档中找到非包容性语言的示例。

## Synopsys 安全注释

作为一家致力于保护和保障客户应用程序安全的组织，Synopsys Software Integrity Group (SIG) 同样致力于客户的数据安全和隐私。本声明旨在为 SIG 客户和潜在客户关于我们的系统、合规性认证、流程和其他安全相关活动的最新信息。

本声明可在以下位置获取：[安全注释 | Synopsys](#)

# 1. 使用 Kubernetes 安装 Black Duck

Kubernetes 是一种编排工具，用于通过容器管理云工作负载。


 警告: 从 Black Duck 2023.7.0 版本开始，Synopsysctl 不再受支持，也不会有更新。Synopsysctl 的文档可在 <https://github.com/blackducksoftware/hub/tree/master/kubernetes/blackduck> 找到。

## 2. 硬件要求

Black Duck 硬件扩展指南

有关可扩展性调整指南，请参阅 [Black Duck 硬件扩展指南](#)。

Black Duck 数据库

 危险：请勿删除 Black Duck 数据库 (bds\_hub) 中的数据，除非 Synopsys 技术支持代表指示这样做。确保遵循适当的备份程序。删除数据会导致 UI 问题、Black Duck 完全无法启动等问题。Synopsys 技术支持无法重新创建已删除的数据。如果没有可用的备份，Synopsys 将尽力提供支持。

磁盘空间要求

所需的磁盘空间量取决于要管理的项目的数量，因此各个要求可能有所不同。考虑每个项目大约需要 200 MB。

Black Duck Software 建议监视 Black Duck 服务器上的磁盘利用率，以防止磁盘达到可能导致 Black Duck 出现问题的容量。

BDBA 扩展

调整 binaryscanner 副本的数量，以及根据每小时将执行的二进制扫描的预期数量增加 PostgreSQL 资源，从而完成 BDBA 扩展。对于每小时每 15 次二进制扫描，添加以下资源：

- 一个 binaryscanner 副本
- 一个用于 PostgreSQL 的 CPU
- 用于 PostgreSQL 的 4GB 内存

如果您的预期扫描速率不是 15 的倍数，则向上舍入。例如，每小时 24 次二进制扫描将需要以下资源：

- 两个 binaryscanner 副本、
- 两个用于 PostgreSQL 的额外 CPU，以及
- 用于 PostgreSQL 的 8GB 额外内存。

当二进制扫描为总扫描量（按扫描计数）的 20% 或更少时，此指南有效。

二进制扫描

如果您获得了二进制扫描的许可，则可能需要增加 uploadcache 容器/Pod 内存，因为这是二进制扫描程序提取和处理二进制文件的位置。默认情况下，内存设置为 512MB，这不足以进行大型扫描。扫描大二进制文件时，建议将 uploadcache 容器/Pod 的内存至少增加到 4 GB。为此，请查找覆盖 yam1 并将内存限制更新为 4096MB。

对于 Swarm 安装：

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



对于 Kubernetes 安装：

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


 注：安装 Black Duck Alert 需要 1 GB 额外内存。

## 3. PostgreSQL 版本


Black Duck 2023.10.0 支持新的 PostgreSQL 特性和功能，以提高 Black Duck 服务的性能和可靠性。从 Black Duck 2023.10.0 开始，PostgreSQL 14 是内部 PostgreSQL 容器支持的 PostgreSQL 版本。

从 Black Duck 2023.10.0 开始，PostgreSQL 设置将在使用 PostgreSQL 容器的部署中自动设置。使用外部 PostgreSQL 的客户仍需手动应用设置。

使用 PostgreSQL 容器并从 2022.2.0 至 2023.7.x (含) 之间的 Black Duck 版本升级的客户，将自动迁移到 PostgreSQL 14。从旧版本 Black Duck 升级的客户需要先升级到 2023.7.x，然后才能升级到 2023.10.0。

 注：有关 PostgreSQL 调整指南，请参阅 [Black Duck 硬件扩展指南](#)。

如果您选择运行自己的外部 PostgreSQL 实例，Synopsys 建议为新安装使用最新版本 PostgreSQL 15。

 警告：不要在 PostgreSQL 数据目录上运行防病毒扫描。防病毒软件会打开大量文件，锁定文件，等等。这些操作会干扰 PostgreSQL 的运行。具体错误因产品而异，但通常会导致 PostgreSQL 无法访问其数据文件。一个例子是，PostgreSQL 失败，并显示“系统中打开的文件太多”。

## 4. 使用 Helm 安装 Black Duck

Helm 图表说明了一组 Kubernetes 资源，Helm 部署 Black Duck 需要用到这些资源。Black Duck 支持 Helm3，Kubernetes 的最低版本为 1.13。

您可以在以下网址获取 Helm 图表：<https://sig-repo.synopsys.com/artifactory/sig-cloudnative>

单击[此处](#)了解有关使用 Helm 安装 Black Duck 的说明。Helm 图表引导在 Kubernetes 群集上使用 Helm 软件包管理器部署 Black Duck。

使用 Helm 在 Kubernetes 上进行迁移

如果您从基于 PostgreSQL 9.6 的 Black Duck 版本升级，此迁移将用 Synopsys 提供的容器替换 CentOS PostgreSQL 容器。此外，synopsys-init 容器将替换为 blackduck-postgres-waiter 容器。

在普通 Kubernetes 上，升级作业的容器将以 root 身份运行（除非覆盖）。但是，唯一的要求是作业与 PostgreSQL 数据卷的所有者以相同的 UID 运行（默认为 UID=26）。

在 OpenShift 上，升级作业假定它将使用与 PostgreSQL 数据卷所有者相同的 UID 运行。

# 5. Artifactory 集成

## 概述


Artifactory 集成是保护软件供应链的 Black Duck 机制。由于 Artifactory 通常是该链的最终环节之一，因此扫描已配置的 Artifactory 存储库组中的每个工件可让客户控制其单个供应链。默认情况下，此版本的 Artifactory 集成会自动阻止从扫描的 Artifactory 存储库中进行违反 Black Duck 策略的下载。定义为“快速”或“两者”的 Black Duck 策略将应用于 Artifactory 集成。

 注：使用 Artifactory 集成会禁用 Artifactory 集成插件的扫描程序和检查器功能。

## 架构

Black Duck 2023.10.0 中改进了 Artifactory 集成的架构方法，以支持重点关注完全托管的部署。这些变化反映在下面的架构图中。

## Artifactory 集成前提条件

 注：您必须在注册密钥上启用 Artifactory 集成，才能使用此功能。启用后，在 values.yaml 文件中添加以下内容：

```
enableIntegration: true
```

类别	要求
Helm 和 Kubernetes	<ul style="list-style-type: none"><li>Kubernetes 1.9+</li><li>Helm3</li><li>将 Synopsys 存储库添加到 Helm 存储库：</li></ul> <pre>\$ helm repo add synopsys https://sig-repo.synopsys.com/artifactory/sig-cloudnative</pre>

调整群集大小 下面的群集大小信息仅供参考，因为具体的托管配置可能有所不同。提供的数据是为了传达支持默认最大工件大小 5GB 所需的大小。

表 1: 核心

	所需核心数	核心总数
1 个管理器	每个管理器 2 个	2
1 个消息处理程序	每个消息处理程序 1 个	1
5 个扫描程序和 BDBA 工作程序	每个扫描程序和 BDBA 工作程序 1 个	5
		8

表 2: 内存

	所需内存	总内存
1 个管理器	每个管理器 4 GB	4 GB

类别	要求															
	<table><tr><th></th><th>所需内存</th><th>总内存</th></tr><tr><td>1 个消息处理程序</td><td>每个消息处理程序 4 GB</td><td>4 GB</td></tr><tr><td>5 个扫描程序</td><td>每个扫描程序 5 GB</td><td>25 GB</td></tr><tr><td>5 个 BDBA 工作程序</td><td>每个 BDBA 工作程序 5 GB</td><td>25 GB</td></tr><tr><td></td><td></td><td>58 GB</td></tr></table>		所需内存	总内存	1 个消息处理程序	每个消息处理程序 4 GB	4 GB	5 个扫描程序	每个扫描程序 5 GB	25 GB	5 个 BDBA 工作程序	每个 BDBA 工作程序 5 GB	25 GB			58 GB
	所需内存	总内存														
1 个消息处理程序	每个消息处理程序 4 GB	4 GB														
5 个扫描程序	每个扫描程序 5 GB	25 GB														
5 个 BDBA 工作程序	每个 BDBA 工作程序 5 GB	25 GB														
		58 GB														
	存储： 托管环境至少包含 100GB 的磁盘空间（扫描程序副本设置为 5）。															
存储类	Artifactory 集成部署需要支持持久卷的、完全调配的存储类。															
持久卷	Artifactory 集成部署需要足够的物理磁盘空间供扫描程序下载工件进行扫描。托管系统将提供至少 100GB 的磁盘空间（扫描程序副本设置为 5）。															
其他要求	Artifactory 集成插件 <ul style="list-style-type: none"><li>Black Duck Artifactory 集成插件安装在目标 JFrog Artifactory Pro 服务器中。</li><li>插件的安装仍然遵循<a href="#">当前说明</a>。请注意特定于 ScanAsAService 模块的插件的配置项目。</li></ul>															

## 安装顺序

下面提供了安装 Artifactory 集成的有序步骤的摘要：

- 从您的 Black Duck 实例获取访问令牌并存储在安全位置。
- 从您的 Artifactory 实例获取访问令牌并存储在安全位置。
- 准备 Kubernetes 环境：
  - 从 sig-cloudnative 提取最新的 Artifactory 集成部署图表。
  - 在 Kubernetes 中为您的部署创建名称空间。
  - 在为 Black Duck 和 Artifactory 访问令牌新创建的名称空间中创建密钥。
- 使用 Artifactory 集成参数编辑 values.yaml。

属性	详细信息
BLACKDUCK_SCA_ENGINE_SCHEME	说明：用于连接到 Artifactory 集成实例的协议。 默认值：不适用 必填：是 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCA_ENGINE_HOST	说明：Artifactory 集成的管理器实例的名称。 默认值：不适用 必填：是 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCA_ENGINE_PORT	说明：Artifactory 集成的管理器实例运行的端口。 默认值：不适用

必填：是  
使用者：  
• scaaas-scanner

- 5. 将 Artifactory 集成安装到名称空间中。
- 6. 准备安装 Artifactory 集成插件：
  - a. 从 GitHub 下载插件。
  - b. 解压缩下载的文件。
  - c. 将插件文件移动到 Artifactory 安装中的相应目录。
- 7. 根据需要编辑 Artifactory 集成插件 blackDuckPlugin.properties 文件。
- 8. 重新启动 Artifactory 服务器。

## 安装 Artifactory 集成

通过 Helm 部署 Artifactory 集成

- 1. 从以下位置下载 Artifactory 集成：
  - Black Duck 2023.4.x 或更早版本的用户，请从 sig-cloudnative 存储库中拉取最新的部署图表。此操作将下载部署存档 (tar.gz)，该存档应解包以用于进一步的部署步骤。

要下载最新图表：

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

要解包并访问图表：

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

- Black Duck 2023.7.x 及更高版本的用户，请从以下位置下载 Artifactory 集成 <https://sig-repo.synopsys.com/artifactory/bds-integrations-release/com/synopsys/integration/artifactory-integration/>
- 2. 如果尚未创建名称空间，请创建名称空间。

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

- 3. 为 Artifactory 和 Black Duck 实例的访问令牌创建密钥。

```
$ 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 访问令牌范围：可以是“用户”，而不是“管理员”，只要满足最低权限（如下所列）：


权限	存储库
<ul style="list-style-type: none"><li>• 读取</li><li>• 注释</li></ul>	需要被 Artifactory 集成扫描的任何本地或远程缓存存储库。

- 
- 部署/缓存 仅扫描报告上传到的“本地”存储库。
- 

## 4. 配置 Artifactory 集成部署。

- 在安装 Artifactory 集成 helm 图表之前，请更新 values.yaml 文件。
- 为以下环境变量设置正确的值（有关更多详细信息，请参阅下面的 [Artifactory 集成应用程序配置部分](#)）。

```
enviros:
  BLACKDUCK_SCAAAS_BLACKDUCK_HOST: ""
  BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME: ""
  BLACKDUCK_SCAAAS_STRUCTURED_LOGGING: ""
  BLACKDUCK_SCA_ENGINE_SCHEME:
  BLACKDUCK_SCA_ENGINE_HOST:
  BLACKDUCK_SCA_ENGINE_PORT:
```

 注：如果上面的列表中有任何未设置的环境变量，建议您通过在其前面添加一个井号 (#) 来排除它，或者将其删除。


- 确保“密钥”部分已更新，以便访问以前创建的密钥：

```
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. 使用 helm 图表部署 Artifactory 集成。

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

## 安装 Artifactory 插件 (已弃用)

 警告：此版本的 Artifactory 插件在 Black Duck 2023.7.0 版本中已弃用。有关 Artifactory 集成插件的当前版本，请参阅“安装 Artifactory 集成插件”。

以下步骤概述了安装和配置 Black Duck Artifactory 插件的过程。

- 从以下位置下载 Artifactory 插件：

<https://github.com/blackducksoftware/blackduck-artifactory/releases>

- 下载并解压缩 artifactory-integration-<version>.[zip | tgz] 文件后，您可以配置插件属性文件。
- 获取一个用于 blackDuckArtifactoryIntegration.properties 文件中的凭据的 Black Duck API 令牌。
- 使用 /plugins/lib 文件夹中的 blackDuckPlugin.properties 文件配置 Black Duck 凭据。

了解有关 [配置 Artifactory 集成](#) 的信息。

- 将 plugins 和 lib 文件夹复制到 \${JFrog\_ARTIFACTORY\_HOME}/etc/plugins/

复制时如下所示：

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

## 5. Artifactory 集成 • 安装 Artifactory 集成插件

### 6. 更改以下文件夹的用户：

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

### 7. 重新启动 Artifactory 服务器。

#### 测试连接

当您安装和配置 Black Duck 插件时，Synopsys 建议您测试连接并确保插件工作正常。使用以下 curl 命令测试连接。

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

#### 使用 Docker 安装的 Artifactory

执行 Docker cp 命令以将插件文件和 lib 文件夹从解压位置移动到 \${ARTIFACTORY\_HOME}/etc/plugins/。

## 安装 Artifactory 集成插件

以下步骤描述了安装和配置 Artifactory 集成插件的过程。

### 1. 从以下位置下载 Artifactory 插件：

<https://sig-repo.synopsys.com/artifactory/bds-integrations-release/com/synopsys/integration/artifactory-integration/>

- 下载并解压缩 artifactory-integration-<version>.[zip | tgz] 文件后，您可以配置插件属性文件。
- 获取 Black Duck API 令牌以用作 blackDuckArtifactoryIntegration.properties 文件中的凭据。
- 使用 blackDuckArtifactoryIntegration.properties 文件配置 Black Duck 凭据，该文件位于 artifactory-integration-<version>/lib 文件夹中。

了解有关[配置插件](#)的信息。

- 将 artifactory-integration-<version>/blackDuckArtifactoryIntegration.groovy 文件和 artifactory-integration-<version>/lib 文件夹复制到 \${ARTIFACTORY\_HOME}/var/etc/plugins/
- 更改以下文件夹的用户：

- ```
chown -R 1030:1030 ${ARTIFACTORY_HOME}/var/etc/plugins/blackDuckArtifactoryIntegration.groovy
```
- ```
chown -R 1030:1030 ${ARTIFACTORY_HOME}/var/etc/plugins/lib
```

### 7. 重新启动 Artifactory 服务器。

#### 使用 Docker 安装的 Artifactory


执行 Docker cp 命令，将插件 groovy 文件和 lib 文件夹从提取的位置移至 \${ARTIFACTORY\_HOME}/var/etc/plugins/。



## 配置 Artifactory 插件 (已弃用)

对于 Black Duck Artifactory 集成插件版本 6.0.0 和更高版本，所有插件及其配置属性都将合并到 blackDuckPlugin.properties 文件中。

必须先修改 blackDuckPlugin.properties 文件，插件才能正常运行，您可以使用任何文本编辑器手动编辑属性文件以配置该插件。您可以编辑任何属性，包括默认值。因此，如果您使用的插件版本低于 6.0.0，强烈建议您完全重新安装。

 注：在 Black Duck Artifactory 集成插件版本 6.0.0 及更高版本中，所有与 Hub 相关的属性将被移除，不再受支持。

下文概述了 blackDuckPlugin.properties 文件的重要设置。

### Black Duck 连接凭据

您需要连接到 Black Duck，可在属性文件中配置此连接。

至少，您必须在属性文件的 BlackDuck 凭据下添加令牌 blackduck.api.token=<BD API token> 和 Black Duck URL。


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

使用的是访问令牌，而不是 Black Duck 代理，那么这就是您在属性文件中的“凭据”部分需要提供的全部信息。

### 扫描即服务设置

最重要的扫描程序设置是存储库列表。您可以定义“扫描即服务”模块确认的存储库名称。如果不调整这些设置，您将无法阻止违反已定义的策略的项目。

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

 注：不支持虚拟存储库，因为它们不包含组件。应配置虚拟存储库引用的本地存储库以进行扫描。

另一个重要的设置是截止日期。如果您有不需扫描的较旧应用程序，此日期将定义截止日期，在该日期之前的应用程序将被忽略。无论阻止策略值如何，lastUpdated 时间早于该值的工件都不受阻止策略的约束。

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

### 属性文件

必须先修改以下属性文件，插件才能正常工作。

```
# 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
```

## 5. Artifactory 集成 • 安装 Artifactory 集成插件

```
# 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
```

## 配置 Artifactory 集成插件

必须先修改 `blackDuckArtifactoryIntegration.properties` 文件，插件才能正常运行，您可以使用任何文本编辑器手动编辑属性文件以配置该插件。

下文概述了 `blackDuckArtifactoryIntegration.properties` 文件的重要设置。

### Black Duck 连接凭据

您需要连接到 Black Duck，可在属性文件中配置此连接。

您必须在属性文件中的 Black Duck 凭据下添加 Black Duck 令牌 `blackduck.api.token=<BD API token>` 和 Black Duck URL。

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

使用的是访问令牌，而不是 Black Duck 代理，那么这就是您在属性文件中的“凭据”部分需要提供的全部信息。

### Artifactory 配置名称

您必须将配置名称设置为与 Black Duck 实例中 [Artifactory 集成配置](#) 的名称一致。Artifactory 集成初始化后，它将连接到上面配置的 Black Duck 实例，并根据以下配置检索集成设置：

```
blackduck.artifactory.config.name=
```

如果 BlackDuck 实例中不存在给定 `blackduck.artifactory.config.name` 的配置，则会记录错误，Artifactory 集成将不会加载到您的 Artifactory 实例中。您需要修改名称并重新启动 Artifactory 实例。

## 测试连接

当您安装和配置 Black Duck 插件时，Synopsys 建议您测试连接并确保插件工作正常。使用以下 `curl` 命令测试连接：

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

## Artifactory 集成任务

### 升级 Artifactory 集成

1. 在升级到新版本之前，请运行以下命令以从图表博物馆中提取最新版本的图表：

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

2. 升级 Artifactory 集成

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

### 更新 Artifactory 集成

更新是一种特定类型的升级，允许对同一版本进行更改，例如添加 ENV 变量。更新时可以使用 `--reuse-values` 标志。

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

### 重新启动 Artifactory 集成

要重新启动 Artifactory 集成服务，可使用以下选项：

1. 将 Pod 缩减至 “0”，然后再恢复为所需的副本数。

要停止：

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

要开始：

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

2. 在 `values.yaml` 中编辑状态。

将状态从 “正在运行” 更改为 “已停止”，然后运行 “helm upgrade”：

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

将状态从 “已停止” 更改为 “正在运行”，然后运行 “helm upgrade”：

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

### 移除 Artifactory 集成

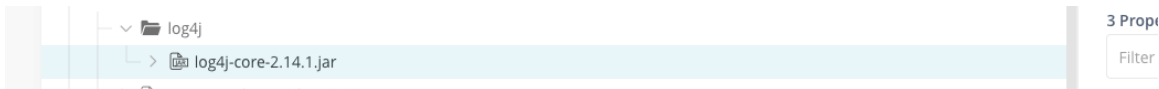
要卸载/删除部署：

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

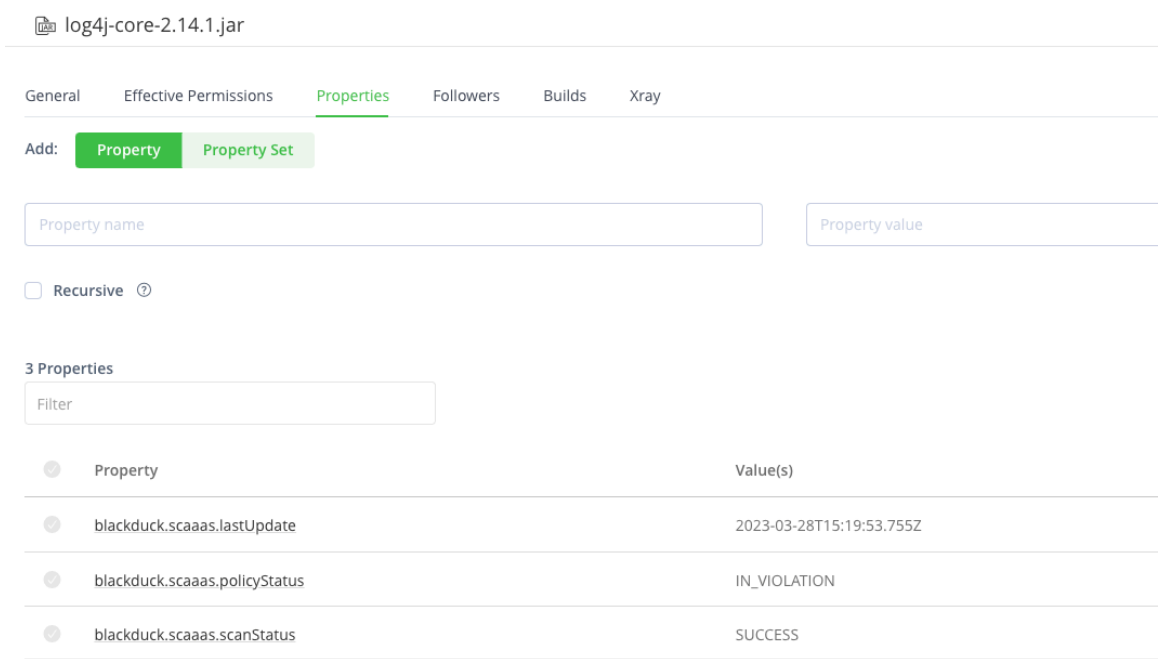
### 手动覆盖阻止的下载

如果 Artifactory 中的项目违反了您在 BlackDuck 中定义的策略，但您希望覆盖并允许下载该项目，请遵循以下说明：

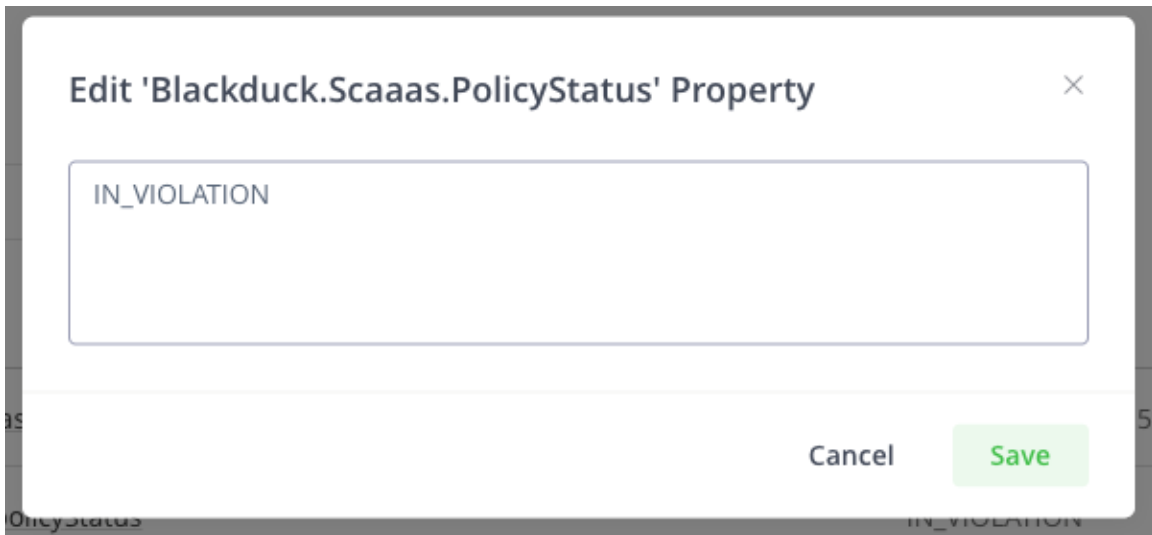
1. 登录 Artifactory UI 并找到您要覆盖的违规项目。



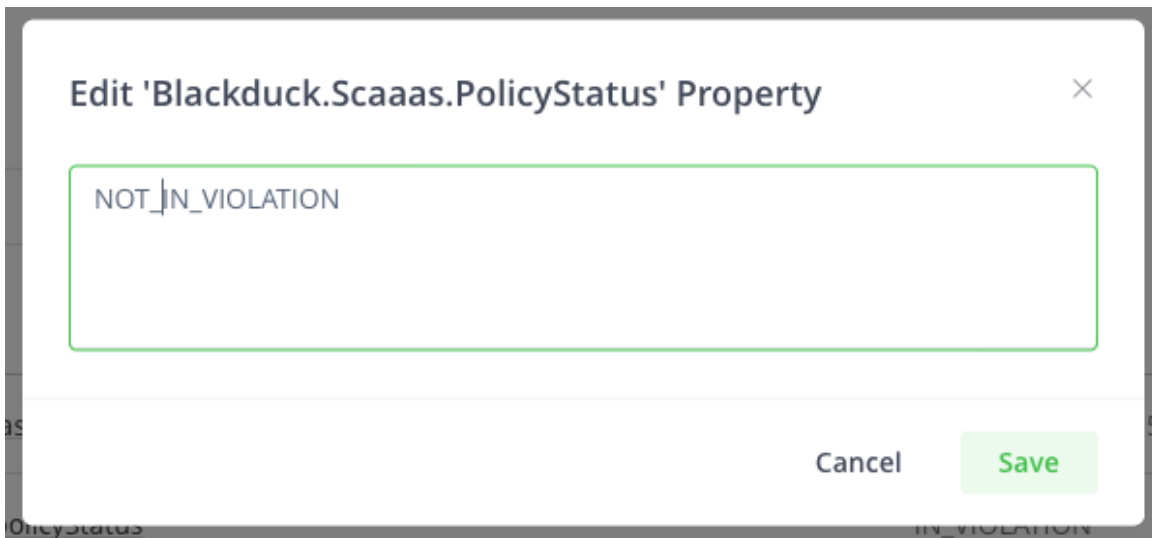
2. 从右窗格中选择“属性”。



3. 选择 `blackduck.scaaas.policyStatus` 属性并将其值修改为 `NOT_IN_VIOLATION`，然后单击“保存”。




Dialog box titled "Edit 'Blackduck.Scaaas.PolicyStatus' Property". The input field contains the text "IN\_VIOLATION". At the bottom right, there are "Cancel" and "Save" buttons. The "Save" button is highlighted in green.



Dialog box titled "Edit 'Blackduck.Scaaas.PolicyStatus' Property". The input field contains the text "NOT\_IN\_VIOLATION". At the bottom right, there are "Cancel" and "Save" buttons. The "Save" button is highlighted in green.

现在，无论 Artifactory 集成插件中设置的阻止策略如何，您都应该能够下载该项目。

 注：如果项目已更新（例如，上传了新版本），它将被重新扫描，并且 `policyStatus` 可以再次设置为 `IN_VIOLATION`。您需要再次执行这些步骤以覆盖并允许下载文件。

#### 禁用二进制和容器扫描

如果您的许可证不允许二进制和容器扫描，请在您的 `values.yaml` 文件中禁用 BDBA。这将导致 `bdbaworker` 容器不会加载，或被卸载（如果容器已加载）。仅支持签名扫描。

要禁用二进制和容器扫描，请编辑 `values.yaml` 文件的“`bdbaworker`”部分并设置：

```
enabled: false
```

保存文件，并按照上述[更新 Artifactory 集成](#)部分中的步骤对部署进行更改。

## 配置 Artifactory 集成

下表列出了 Artifactory 集成图表的可配置参数及其默认值。

Artifactory 集成应用程序配置

下表介绍了 Artifactory 集成的配置项目：

属性名称	详细信息
BLACKDUCK_RABBIT_SSL	说明：将 SSL 用于 rabbit 通信。 默认值：false 必填：否 使用者： <ul style="list-style-type: none"> <li>scaaas-manager</li> <li>scaaas-scanner</li> <li>rabbitmq</li> </ul>
BLACKDUCK_SCAAAS_BLACKDUCK_HOST	说明：Blackduck 实例的名称。 默认值：不适用 必填：是 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_BLACKDUCK_PORT	说明：运行 Black Duck 实例的端口。 默认值：不适用 必填：是 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_BLACKDUCK_SCHEME	说明：用于连接到 Blackduck 实例的协议。 默认值：不适用 必填：是 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_BLACKDUCK_TOKEN	说明：用于扫描身份验证的 Blackduck API 令牌，仅限开发部署。 默认值：不适用 必填：否 使用者： <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_COMPRESS_ALL_MQ_MESSAGES	说明：启用后，将压缩所有 rabbitMQ 消息。 默认值：false 必填：否 使用者： <ul style="list-style-type: none"> <li>scaaas-manager</li> <li>scaaas-scanner</li> </ul>

属性名称	详细信息
BLACKDUCK_SCAAAS_DETECT_BDBA_TIMEOUT	<p>说明：Detect BDBA 扫描超时（以秒为单位）。</p> <p>默认值：3600</p> <p>必填：否</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_DETECT_TIMEOUT	<p>说明：Detect 扫描超时（以秒为单位）。</p> <p>默认值：600</p> <p>必填：否</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_DOWNLOAD_TIMEOUT_MINUTES	<p>说明：如果扫描程序下载在异常情况下结束，并且花费的时间超过此超时时间，则该项目将被标记为“失败”。</p> <p>默认值：30 分钟</p> <p>必填：否</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
BLACKDUCK_SCAAAS_STALL_ON_FAILURE	<p>说明：通过在后台运行 Java 并对其进行监视，我们可以选择在 Java 出现启动问题或崩溃时停止运行。启用此选项时，它仍然能够连接到容器以进行调试和分析。</p> <p>默认值：false</p> <p>必填：否</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-manager</li> <li>scaaas-scanner</li> </ul>
DETECT_LATEST_RELEASE_VERSION	<p>说明：要用于扫描的 Detect 版本，Artifactory 集成需要 Detect 8.2.0 及更高版本。</p> <p>默认值：&lt;value as configured from synopsys-detect dependency&gt;</p> <p>必填：否</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-scanner</li> </ul>
RABBIT_MQ_PORT	<p>说明：用于 rabbitMQ 通信的端口。</p> <p>默认值：5672</p> <p>必填：是</p> <p>使用者：</p> <ul style="list-style-type: none"> <li>scaaas-manager</li> <li>scaaas-scanner</li> <li>rabbitmq</li> </ul>




## 6. 管理任务

### 在 Kubernetes 中配置密钥加密

Black Duck 支持对系统中的关键数据进行静态加密。此加密基于编排环境（Docker Swarm 或 Kubernetes）调配给 Black Duck 安装的密钥。创建和管理此密钥、创建备份密钥以及根据您所在组织的安全策略轮换密钥的过程如下所述。

要加密的关键数据如下：

- SCM 集成 OAuth 令牌
- SCM 集成提供商 OAuth 应用程序客户端密钥
- LDAP 凭据
- SAML 私有签名证书

 注：一旦启用了密钥加密，就永远不能禁用它。

什么是加密密钥？

加密密钥是一个随机序列，用于生成内部加密密钥以解锁系统内的资源。Black Duck 中的密钥加密由 3 个对称密钥（根密钥、备份密钥和以前的密钥）控制。这三个密钥通过传递到 Black Duck 的种子，作为 Kubernetes 和 Docker Swarm 密钥生成。这三个密钥被命名：

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

在正常情况下，并非全部三个种子都会被激活使用。除非正在执行轮换操作，否则唯一活动的种子将是根种子。

保护根种子

必须保护根种子。拥有您的根种子以及系统数据副本的用户可以解锁并读取系统的受保护内容。某些 Docker Swarm 或 Kubernetes 系统默认情况下不静态加密密钥。强烈建议将这些编排系统配置为在内部加密，以便以后在系统中创建的密钥能够保持安全。

根种子是从备份重新创建系统状态（作为灾难恢复计划的一部分）所必需的。根种子文件的副本应存储在独立于编排系统的秘密位置，以便种子与备份的组合可以重新创建系统。不建议将根种子存储在与备份文件相同的位置。如果一组文件被泄露或被盗 – 两种情况都会出现，因此，建议为备份数据和种子备份设置单独的位置。

在 Kubernetes 中启用密钥加密

要在 Kubernetes 中启用密钥加密，必须在 values.yaml 编排文件中将 enableApplicationLevelEncryption 的值更改为 true：

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

## 6. 管理任务 • 在 Kubernetes 中生成种子

### 密钥种子管理脚本

您可以在 Black Duck GitHub 公共存储库中找到示例管理脚本：

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

这些脚本不是用来管理 Black Duck 密钥加密，而是用来说明此处所述的低级 Docker 和 Kubernetes 命令的用法。有两组脚本，每组都在其自己的子目录中，对应于在 Kubernetes 和 Docker Swarm 平台上使用。对于 Kubernetes 和 Docker Swarm，各个脚本之间存在一对一对应关系（如果适用）。例如，两组脚本都包含一个具有如下名称的脚本：

createInitialSeeds.sh

## 在 Kubernetes 中生成种子

### 在 OpenSSL 中生成种子

可以使用任何机制（生成至少 1024 字节长度的安全随机内容）生成种子的内容。一旦创建种子并将其保存在密钥中，就应将其从文件系统中移除并保存在一个私密的安全位置。

OpenSSL 命令如下所示：

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

### 在 Kubernetes 中生成种子

有许多 Kubernetes 命令行将创建密钥。下面列出的命令可以更好地跟踪密钥及其是否更改，并确保能够使用联机系统操纵密钥。在 Black Duck 激活运行时，密钥可以在 Kubernetes 中创建和删除。

```
kubect1 create secret generic crypto-root-seed -n $NAMESPACE --save-config --dry-run=client --  
from-file=crypto-root-seed=./root_seed -o yaml | kubect1 apply -f -
```

要删除 Kubernetes 中之前的密钥：

```
kubect1 delete secret crypto-prev-seed -n $NAMESPACE
```

## 配置备份种子

建议备份根种子，以确保系统可以在灾难恢复场景中恢复。备份根种子是一个备用根种子，可用于恢复系统。因此，它必须以与根种子相同的方式安全地存储。

备份根种子具有一些特殊特性，即，一旦它与系统关联，即使在根种子轮换期间，它也仍然可行。一旦系统处理了备份种子，应将其从密钥中移除，以限制其受到攻击和泄漏的可能性。备份根种子可能有不同的（频率较低的）轮换计划，因为系统中的密钥不应在任何时候都处于“活动”状态。

当您需要或想要轮换根种子时，首先需要将当前根种子定义为上一个根种子。然后，您可以生成一个新的根种子并将其放置到位。

当系统处理这些种子时，以前的根密钥将用于轮换资源，以使用新的根种子。完成此处理后，应从密钥中移除之前的根种子，以完成轮换并清理旧资源。

### 创建备份根种子

初始创建后，备份种子/密钥将 TDEK（租户解密、加密密钥）低级密钥打包。示例脚本 createInitialSeeds.sh 将创建根种子和备份种子。一旦 Black Duck 运行，它使用两个密钥来打包 TDEK。

该操作完成并且根种子和备份种子都安全地存储在其他位置后，应删除备份种子密钥；请参阅[示例脚本 cleanupBackupSeed.sh](#)。

如果根密钥丢失或泄漏，备份密钥可用于替换根密钥；请参阅[示例脚本 useRootSeed.sh](#)。

### 轮换备份种子

与根密钥类似，备份种子应定期轮换。与根种子不同（旧的根种子存储为以前的种子密钥，而新的根种子密钥提供给系统），备份种子只是通过创建新的备份种子来进行轮换。请参阅[示例脚本 rotateBackupSeed.sh](#)。

轮换完成后，新的备份种子应安全存储并从 Black Duck 主机文件系统中移除。

## 在 Kubernetes 中管理密钥轮换

根据组织的安全策略定期轮换正在使用的根种子是一种好做法。要执行此操作，还需要一个额外的密钥来执行轮换。要轮换根种子，将当前根种子配置为“上一个根种子”，并生成新生成的根种子并将其配置为根种子。一旦系统处理此配置（具体细节如下），密钥将被轮换。

此时，新旧种子都能够解锁系统内容。默认情况下，将使用新的根种子，允许您测试并确保系统按预期工作。一旦所有内容都得到验证，您就可以通过移除“以前的根种子”来完成轮换。

从系统中移除之前的根种子后，就不能再将其用于解锁系统内容，并且可以将其丢弃。新的根种子现在是正确的根种子，应适当地备份和保护。

根密钥用于打包实际加密和解密 Black Duck 密钥的低级 TDEK（租户解密、加密密钥）。应该在方便 Black Duck 管理员并符合用户组织规则时，定期轮换根密钥。

轮换根密钥的过程是使用当前根种子的内容创建以前的种子密钥。然后，应创建一个新的根种子并将其存储在根种子密钥中。

### Kubernetes 中的密钥轮换

对于 Kubernetes 来说，这三个操作都可以在运行 Black Duck 的情况下完成。Kubernetes 示例脚本 `rotateRootSeed.sh` 将把根种子提取到 `prev_root` 中，创建一个新的根种子，然后重新创建以前的种子和根种子。

轮换完成后，应移除上一个种子密钥；请参阅[示例脚本 cleanupPreviousSeed.sh](#)。同样，可以对正在运行的 Kubernetes Black Duck 实例执行此清理。

在用户界面中，转到“管理”>“系统信息”>“加密”，查看“加密诊断”选项卡即可跟踪轮换状态。

## 为 Blackduck 存储配置自定义卷

存储容器可配置为最多使用三 (3) 个卷来存储基于文件的对象。此外，可以将配置设置为将对象从一个卷迁移到另一个卷。

### 为什么使用多个卷？

默认情况下，存储容器使用单个卷来存储所有对象。此卷的大小取决于存储对象的典型客户使用情况。由于每个客户都不同，因此可能需要拥有比卷所能提供的空间更多的可用空间。由于并非所有卷都是可扩展的，因此可能需要添加不同的、更大的卷并将数据迁移到新卷。

可能需要多个卷的另一个原因是：卷托管在远程系统（NAS 或 SAN）上，并且该远程系统将被停用。需要创建托管在新系统上的第二个卷，并将内容移至该卷。

配置多个卷

要在 Kubernetes 中配置自定义存储提供商，请创建包含以下内容的覆盖文件：

```
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"
```

在上述覆盖文件中，提供商 1 和 提供商 2 均已启用，而提供商 2 具有较高的优先级（较低的偏好程度编号），因此所有新内容都将定向到该位置。

每个提供商的可能设置如下所示：

设置	详细信息
name	默认值：无。 有效值：任意。 备注：这是一个识别标签，用于帮助管理这些提供商。
enabled	默认值：true用于提供商 1，false 用于其他。 有效值：true 或 false。 备注：指示是否启用提供商。
index	默认值：无。 有效值：1、2、3。 备注：指示提供商编号。配置文件中的顺序不重要。
type	默认值：file。 有效值：file。

设置	详细信息
	备注：“file”是唯一支持的提供商类型。
preference	<p>默认值：index乘以 10。</p> <p>有效值：0-999。</p> <p>备注：设置提供商的偏好程度。有最高优先级（最低偏好程度编号）的提供商将具有向其添加的新内容。</p> <p>注意：所有提供商偏好程度必须唯一，两个提供商不能共享相同的值。</p>
readonly	<p>默认值：false。</p> <p>有效值：true 或 false。</p> <p>备注：表示提供商为只读。最高优先级（最低偏好程度编号）的提供商不能为只读状态，否则系统无法正常工作。</p> <p>处于“只读”状态的提供商不会因添加数据或移除数据而更改存储卷，但数据库中的元数据将被处理，以记录对象删除和其他更改。</p>
migrationMode	<p>默认值：none。</p> <p>有效值：none、drain、delete、duplicate。</p> <p>备注：配置提供商的迁移模式。本文档的迁移章节详细介绍了此模式以及使用它的方法。</p>
existingPersistentVolumeClaimName	<p>默认值：""。</p> <p>有效值：任何有效的 k8s 标识符。</p> <p>备注：允许您为该卷指定特定的持久性卷声明名称。</p>
pvc.size	<p>默认值：none。</p> <p>有效值：任何有效大小。</p> <p>备注：允许您指定该卷的可用空间量。</p>
pvc.storageClass	<p>默认值：""。</p> <p>有效值：任何有效的 k8s 标识符。</p> <p>备注：允许您为该卷指定特定的存储类。</p>
pvc.existingPersistentVolumeName	<p>默认值：""。</p> <p>有效值：任何有效的 k8s 标识符。</p> <p>备注：允许您为该卷指定特定的持久性卷名称。</p>
mountPath	<p>默认值：特定于索引 - 请参阅注释。</p> <p>有效值：</p> <p>/opt/blackduck/hub/uploads</p> <p>/opt/blackduck/hub/uploads2</p> <p>/opt/blackduck/hub/uploads3</p> <p>备注：</p> <p>设置特定提供商的挂载点。索引为一 (1) 的提供商必须指定挂载点</p> <p>/opt/blackduck/hub/uploads。索引为二 (2) 的提供商必须指定挂载点</p> <p>/opt/blackduck/hub/uploads2。索引为三 (3) 的提供商必须指定挂载点</p> <p>/opt/blackduck/hub/uploads3</p>

在卷之间迁移

配置多个卷后，可以将内容从一个或多个提供商卷迁移到新的提供商卷。这只能对不是最高优先级（最低偏好程度）的提供商执行。为此，请使用以下迁移模式之一配置卷。配置完成后，需要重新启动 Black Duck 才能启动迁移，迁移由后台作业执行，直至完成。

迁移模式	详细信息
none	目的：表示没有正在进行的迁移。 备注：默认迁移模式。
drain	目的：此模式将内容从配置的提供商移动到最高优先级（最低偏好程度编号）的提供商。移动内容后，将立即从源提供商中移除该内容。 备注：这是一个直接的移动操作 - 将其添加到目标提供商并从来源中移除。
delete	目的：此模式将内容从配置的提供商复制到最高优先级（最低偏好程度编号）的提供商。复制内容后，该内容将在源提供商中标记为删除。应用标准删除保留期 - 在该期限之后，内容将被移除。 备注：此移动允许系统在保留窗口期内从备份中恢复，以便源提供商中的内容仍然保持可行。默认保留期为 6 小时。
duplicate	目的：此模式将内容从配置的提供商复制到最高优先级（最低偏好程度编号）的提供商。复制内容后，来源（包括元数据）将保持不变。 备注：重复迁移后，数据库中将有多个卷，其中包含所有内容和元数据。如果您在“复制和转储”过程中执行下一步骤并取消配置原始卷，则文件将被删除，但元数据将保留在数据库中 - 引用未知卷，并在删减程序作业中生成警告（作业错误）。要解决此错误，请使用以下属性启用孤立元数据记录的删减： <code>storage.pruner.orphaned.data.pruning.enable=true</code>

配置 jobrunner 线程池

在 Black Duck 中，有两个作业池 - 一个运行计划作业（称为定期池），另一个运行从某些事件（包括 API 或用户交互）启动的作业（称为按需池）。

每个池都有两个设置：最大线程数和预取。

最大线程数是 jobrunner 容器可以同时运行的最大作业数。将定期和按需最大线程数相加，总和不应大于 32，因为大多数作业使用数据库，并且最多有 32 个连接。Jobrunner 内存很容易饱和，因此默认的线程数设置得非常低。

预取是每个 jobrunner 容器在每次往返数据库的过程中将抓取的作业数。该值设置得越高，效率越高，但该值设置得越低，将使负载更均匀地分布在多个 jobrunner 中（但通常情况下，均匀的负载不是 jobrunner 的设计目标）。

在 Kubernetes 中，可以使用以下覆盖文件覆盖线程计数设置：

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

```
maxOndemandPrefetch: 2
```