



使用 Kubernetes 和 OpenShift 安装 Black Duck SCA

Black Duck SCA 2025.7.0

Black Duck 版权所有 ©2025。

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是其各自所有者的专有财产。

04-11-2025

内容

| | |
|-------------------------------------------------|----|
| 前言..... | 5 |
| Black Duck 文档..... | 5 |
| 客户支持..... | 5 |
| Black Duck 社区..... | 6 |
| 培训..... | 6 |
| Black Duck 关于包容性和多样性的声明..... | 6 |
| Black Duck 安全承诺..... | 6 |
| 1. 使用 Kubernetes 和 OpenShift 安装 Black Duck..... | 8 |
| 2. 硬件要求..... | 9 |
| 3. PostgreSQL 版本..... | 10 |
| 常规迁移过程..... | 10 |
| 4. 使用 Helm 安装 Black Duck..... | 11 |
| Black Duck helm 图表..... | 11 |
| 入门..... | 11 |
| 配置您的 Black Duck 实例..... | 12 |
| 公开 Black Duck 用户界面..... | 13 |
| 安装图表..... | 14 |
| 卸载图表..... | 14 |
| 升级图表..... | 15 |
| 配置参数..... | 15 |
| 常用配置..... | 16 |
| 身份验证 Pod 配置..... | 18 |
| 二进制扫描程序 Pod 配置..... | 18 |
| BOM 引擎 Pod 配置..... | 19 |
| CFSSL Pod 配置..... | 19 |
| Datadog Pod 配置..... | 20 |
| 文档 Pod 配置..... | 20 |
| 集成 Pod 配置..... | 20 |
| 作业运行器 Pod 配置..... | 21 |
| Logstash Pod 配置..... | 21 |
| 匹配引擎 Pod 配置..... | 22 |
| PostgreSQL Pod 配置..... | 22 |
| PostgreSQL 就绪性初始化容器配置..... | 24 |
| PostgreSQL 升级作业配置..... | 24 |
| RabbitMQ Pod 配置..... | 24 |
| Redis Pod 配置..... | 24 |
| 注册 Pod 配置..... | 25 |
| 扫描 pod 配置..... | 25 |
| 存储 Pod 配置..... | 26 |
| Webapp Pod 配置..... | 27 |

| | |
|----------------------------------|----|
| Webserver Pod 配置..... | 28 |
| 5. 管理任务..... | 29 |
| 在 Kubernetes 中配置密钥加密..... | 29 |
| 在 Kubernetes 中生成种子..... | 30 |
| 配置备份种子..... | 30 |
| 在 Kubernetes 中管理密钥轮换..... | 31 |
| 通过 Kubernetes 密钥传递外部数据库凭据..... | 31 |
| 为 Blackduck 存储配置自定义卷..... | 32 |
| 配置 jobrunner 线程池..... | 35 |
| 配置就绪探针..... | 35 |
| 配置 HUB_MAX_MEMORY 设置..... | 35 |
| 使用 Helm 在 OpenShift 上进行迁移..... | 36 |
| 预置 JWT 公钥/私钥对..... | 36 |
| 启用 SCM 集成..... | 36 |
| 在外部数据库中配置 mTLS..... | 37 |
| 从 Synopsysctl 转换到 Helm 图表部署..... | 38 |

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.blackduck.com/bundle/detect/page/integrations/integrations.html>
- https://documentation.blackduck.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 Black Duck 客户支持：

- 在线：<https://community.blackduck.com/s/contactsupport>
- 要创建支持案例，请登录 Black Duck Community 网站：<https://community.blackduck.com/s/contactsupport>。
- 另一个可随时使用的方便资源是[在线社区门户](#)。

Black Duck 社区

Black Duck 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 Black Duck 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 Black Duck 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 Black Duck 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 Black Duck 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 Black Duck 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@blackduck.com。

培训


Black Duck “客户教育”是满足您所有 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 Black Duck 教育，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://blackduck.skilljar.com/page/black-duck>，或者，要获取 Black Duck 的帮助

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

Black Duck 关于包容性和多样性的声明

Black Duck 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。

Black Duck 安全承诺

作为一家致力于保护和保障客户应用程序安全的组织，Black Duck 同样致力于客户的数据安全和隐私。本声明旨在为 Black Duck 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：[安全承诺 | Black Duck](#)

1. 使用 Kubernetes 和 OpenShift 安装 Black Duck


Kubernetes 和 **OpenShift™** 是用于通过容器管理云工作负载的编排工具。Black Duck 通过 Helm 图表支持部署，请参阅 `%install dir%/kubernetes/blackduck/` 中的文档以及 Helm 图表示例。

2. 硬件要求

支持的系统

Black Duck 支持安装和运行以下系统：


- 64 位 x86
- ARM64 (AArch64)

 注：ARM 系统目前不支持 BDBA 和 RL 服务。

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 Black Duck 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。Black Duck 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，Black Duck 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 软件建议监视 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。


 注：安装 Black Duck Alert 需要 1 GB 的额外内存。

3. PostgreSQL 版本


Black Duck 2024.10.0 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 2024.10.0 开始，PostgreSQL 15 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。

Black Duck 2024.10.0 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 2024.10.0 开始，PostgreSQL 15 是内部 PostgreSQL 容器支持的 PostgreSQL 版本。

 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，Black Duck 建议为新安装使用最新版本 PostgreSQL 16。

 注：Black Duck 2025.4.0 增加了对使用 PostgreSQL 17 作为外部数据库的初步支持，仅用于测试；对 PostgreSQL 17 的仅测试用途支持将在 2025.7.0 中进一步延长。

 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”。

常规迁移过程

此处的指南适用于从任何基于 PG 9.6 的 Hub（早于 2022.2.0 的版本）升级到 2022.10.0 或更高版本。

1. 迁移由 blackduck-postgres-upgrader 容器执行。
2. 如果从基于 PostgreSQL 9.6 的 Black Duck 版本升级：
 - PostgreSQL 数据卷的文件夹布局经过重新排列，使未来的 PostgreSQL 版本升级更加简单。
 - 数据卷所有者的 UID 已更改。新的默认 UID 为 1001，但请参见特定于部署的说明。
3. 运行 pg_upgrade 脚本以将数据库迁移到 PostgreSQL 13。
4. 在 PostgreSQL 13 数据库上运行普通“分析”以初始化查询计划程序统计信息。
5. blackduck-postgres-upgrader 退出。

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 Helm 3.5.4，Kubernetes 最低版本为 1.17。

Helm 图表可在此处获取：<https://repo.blackduck.com/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 Black Duck 提供的容器替换 CentOS PostgreSQL 容器。此外，blackduck-init 容器会被替换为 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。

Black Duck helm 图表

此图表使用 Helm 软件包管理器引导 Kubernetes 集群上的 Black Duck 部署。

 注：本文档介绍了安装基本部署的快速启动过程。有关更多配置选项，请参阅 Kubernetes 文档。

前提条件

- Kubernetes 1.16+
 - 配置了允许持久卷的 storageClass。

使用中的 storageClass 的 reclaimPolicy 应设置为 Retain，以确保数据的持久性。AzureFile（非 CSI 变体）需要一个用于 RabbitMQ 的自定义存储类，因为它被视为 SMB 挂载，一旦挂载到 Pod 中，文件和目录权限将不可变。
- Helm 3
- 将存储库添加到本地 Helm 存储库：

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

入门

本地保存图表

要在计算机上保存图表，请运行以下命令：

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

这会将图表解压缩到指定文件夹（如上述命令中的 -d 标志所示），其中包含部署应用程序所需的文件。

4. 使用 Helm 安装 Black Duck • 配置您的 Black Duck 实例

创建名称空间

通过运行以下命令创建名称空间。该示例使用 bd，但也可以选择任意名称代替。

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


创建自定义 TLS 密钥（可选）

在安装 Black Duck Helm 图表之前，通常会提供一个自定义 Web 服务器 TLS 密钥。使用以下命令创建密钥：

```
$ 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
```

接下来，更新 values.yaml 中的 TLS certificate for Black Duck 块，确保取消注释 tlsCertSecretName 值（需要 tls.crt 和 tls.key 文件）。如果未提供 tlsCertSecretName 值，Black Duck 将生成自己的证书。

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


 注：如果从应用程序上游处理 TLS 终止（即通过入口资源），则不需要此步骤。

配置您的 Black Duck 实例

选择合适的部署规模

Black Duck 提供了几个[Black Duck预配置](#)的每小时扫描次数 yaml 文件，以帮助您适当调整部署规模。我们的性能实验室已使用实际配置对这些文件进行了测试。然而，它们并非“一刀切”，因此，如果您计划运行大量 BDBA 扫描、代码段扫描或报告，请联系您的 CSM，协助您确定自定义规模层级。

自 2024.4.x 版本起，应使用 GEN04 规模文件。

 注：10sph.yaml 文件不适用于生产目的，不应部署用于本地测试之外的任何用途。

配置持久化存储

Black Duck 需要将特定数据持久化存储到磁盘。因此，应在安装中使用适当的 storageClass。如果您的群集没有默认 storageClass，或者您希望覆盖它，请更新以下参数：

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

使用中的 storageClass 的 reclaimPolicy 应设置为 Retain，以确保数据的持久性。AzureFile（非 CSI 变体）需要一个用于 RabbitMQ 的自定义存储类，因为它被视为 SMB 挂载，一旦挂载到 Pod 中，文件和目录权限将不可变。


配置数据库

如果选择使用外部 postgres 实例（默认配置），则需要在 values.yaml 中配置以下参数：

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

如果您选择使用容器化 PostgreSQL 实例，请将以下参数设置为 false：

```
postgres.isExternal: true
```

 注：数据库部署的规格必须符合适当的规模层级。无论选择哪种数据库部署方法，都要确保定期执行备份并定期验证这些备份的完整性。

公开 Black Duck 用户界面

Black Duck 用户界面 (UI) 可以通过多种方法访问，如下所述。

NodePort

NodePort 是 values.yaml 中设置的默认服务类型。如果您想使用自定义 NodePort，请将 value.yaml 中的以下参数设置为所需的端口：

```
# 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>"
```

您可以通过 `https://{NODE_IP}:{NODE_PORT}` 访问 Black Duck 用户界面。

负载均衡器

通过在 values.yaml 中将 exposedServiceType 设置为 LoadBalancer，将指示 Kubernetes 部署外部负载均衡器服务。您可以使用以下命令获取 Black Duck Web 服务器的外部 IP 地址：

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

如果外部 IP 地址显示为待定，请稍等片刻，然后再次输入相同的命令。

您可以通过 `https://{EXTERNAL_IP}` 访问 Black Duck UI。

入口

这通常是向用户公开应用程序的最常用方法。首先，将 values.yaml 中的 exposeui 设置为 false，因为入口将路由到服务。

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

典型的入口清单如以下示例所示。请注意，入口控制器和 TLS 证书本身的配置超出了本指南的范围。

```
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
```

4. 使用 Helm 安装 Black Duck • 安装图表

```
port:
  number: 443
ingressClassName: nginx
```

部署完成后，用户界面将通过入口控制器公有 IP 的 443 端口提供服务。

OpenShift

通过在 values.yaml 中将 exposedServiceType 设置为 OpenShift，将指示 OpenShift 部署路由服务。

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

您可以使用以下命令获取 OpenShift 路由：

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

您可以通过 `https://${ROUTE_HOST}` 访问 Black Duck 用户界面。

安装图表

要安装 Black Duck 图表，请运行以下命令：

```
$ 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
```

 提示：使用 `helm list` 列出所有版本，使用 `helm get values RELEASE_NAME` 列出所有指定值。

安装 Helm 图表时不得使用 `--wait` 标志。在将安装标记为完成之前，`--wait` 标志会等待所有 Pod 进入就绪状态。然而，在安装后期间运行 `postgres-init` 作业之前，Pod 不会进入就绪状态。因此安装永远不会完成。

或者，也可以使用 `kubectl apply` 通过从 Helm 生成模拟运行清单来部署 Black Duck：

```
$ 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
```

卸载图表

要卸载/删除部署：

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

该命令会移除与图表相关联的所有 Kubernetes 组件，并删除版本。

如果如上所示使用 `kubectl` 从模拟运行中进行安装，下面的命令将移除安装：

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

升级图表

在升级到新版本之前，请确保运行以下命令以从图表博物馆中拉取最新版本的图表：

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

要更新部署：

```
$ 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
```

配置参数

下表列出了 Black Duck 图表的可配置参数及其默认值。

- [常用配置](#)
- [身份验证 Pod](#)
- [二进制扫描程序 Pod](#)
- [BOM 引擎 Pod](#)
- [CFSSL Pod](#)
- [Datadog Pod](#)
- [集成 Pod](#)
- [作业运行器 Pod](#)
- [Logstash Pod](#)
- [匹配引擎 Pod](#)
- [PostgreSQL Pod](#)
- [PostgreSQL 就绪性初始化容器](#)
- [PostgreSQL 升级作业](#)
- [RabbitMQ Pod](#)
- [Redis Pod](#)
- [注册 Pod](#)
- [扫描 Pod](#)
- [存储 Pod](#)
- [Webapp Pod](#)
- [Webserver Pod](#)

 注：不要在环境标志中设置以下参数。而应该使用它们各自的标志。

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

4. 使用 Helm 安装 Black Duck • 配置参数

```
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
```

常用配置

| 参数 | 说明 | 默认 |
|-------------------------|--------------------------------------------------|-----------------------------|
| registry | 映像存储库 | docker.io/blackducksoftware |
| imageTag | Black Duck 版本 | 2024.7.1 |
| imagePullSecrets | 拉取映像时使用的一个或多个密钥的引用 | [] |
| tlsCertSecretName | 包含证书的 Webserver TLS 密钥的名称（如果未提供，将生成证书） | |
| exposeui | 启用 Black Duck Web 服务器用户界面 (UI) | true |
| exposedServiceType | 公开 Black Duck Web 服务器服务类型 | NodePort |
| enablePersistentStorage | 如果为 true，Black Duck 将拥有持久性存储 | true |
| storageClass | 所有持久性卷声明中使用的全局存储类 | |
| enableLivenessProbe | 如果为 true，Black Duck 将拥有存活探针 | true |
| enableInitContainer | 如果为 true，Black Duck 将初始化所需的数据库 | true |
| enableSourceCodeUpload | 如果为 true，将通过环境变量中的设置来启用源代码上传（这优先于环境标志值） | false |
| dataRetentionInDays | 源代码上传的数据保留天数 | 180 |
| maxTotalSourceSizeinMB | 源代码上传的最大总源代码大小（以 MB 为单位） | 4000 |
| enableBinaryScanner | 如果为 true，将通过部署二进制扫描工作程序启用二进制分析 | false |
| enableIntegration | 如果为 true，将通过环境变量中的设置来启用 Black Duck 集成（这优先于环境标志值） | false |

| 参数 | 说明 | 默认 |
|--------------------------|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enableAlert | 如果为 true，Black Duck Alert 服务将通过环境变量添加到 nginx 配置中 "HUB_ALERT_HOST:<blackduck_name>--alert.<blackduck_name>.svc" | false |
| enableIPV6 | 如果为 true，将通过环境变量中的设置来启用 IPV6 支持（这优先于环境标志值） | true |
| certAuthCACertSecretName | 用于 Black Duck 证书认证的自有证书颁发机构 (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 代理服务器的证书颁发机构 (CA) | <pre>run this command "kubectl create secret generic -n <namespace> <name>-blackduck- proxy-certificate --from- file=HUB_PROXY_CERT_FILE=proxy.crt" and provide the secret name</pre> |
| proxyPasswordSecretName | Black Duck 代理密码密钥 | <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 密码密钥 | <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 | 需要添加到 Black Duck 配置中的环境变量 | <pre>map e.g. if you want to set PUBLIC_HUB_WEBSERVER_PORT, then it should be --set</pre> |

| 参数 | 说明 | 默认 |
|----|----|-------------------------------------------------------|
| | | <code>enviorns.PUBLIC_HUB_WEBSERVER_PORT=30269</code> |

身份验证 Pod 配置

| 参数 | 说明 | 默认 |
|-------------------------------------------------------|----------------------|-----------------|
| <code>authentication.registry</code> | 要在容器级别覆盖的映像存储库 | |
| <code>authentication.resources.limits.memory</code> | 身份验证容器内存限制 | 1024Mi |
| <code>authentication.resources.requests.memory</code> | 身份验证容器内存请求 | 1024Mi |
| <code>authentication.maxRamPercentage</code> | 身份验证容器最大堆大小 | 90 |
| <code>authentication.persistentVolumeClaimName</code> | 指向现有的身份验证持久卷声明 (PVC) | |
| <code>authentication.claimSize</code> | 身份验证持久卷声明 (PVC) 声明大小 | 2Gi |
| <code>authentication.storageClass</code> | 身份验证持久卷声明 (PVC) 存储类 | |
| <code>authentication.volumeName</code> | 指向现有的身份验证持久卷 (PV) | |
| <code>authentication.nodeSelector</code> | 用于 Pod 分配的身份验证节点标签 | <code>{}</code> |
| <code>authentication.tolerations</code> | 用于 Pod 分配的身份验证节点容忍度 | <code>[]</code> |
| <code>authentication.affinity</code> | 用于 Pod 分配的身份验证节点亲和性 | <code>{}</code> |
| <code>authentication.podSecurityContext</code> | Pod 级别的身份验证安全上下文 | <code>{}</code> |
| <code>authentication.securityContext</code> | 容器级别的身份验证安全上下文 | <code>{}</code> |

二进制扫描程序 Pod 配置

| 参数 | 说明 | 默认 |
|------------------------------------------------------|------------------------|-------------------------------------|
| <code>binaryscanner.registry</code> | 要在容器级别覆盖的映像存储库 | <code>docker.io/sigblackduck</code> |
| <code>binaryscanner.imageTag</code> | 要在容器级别覆盖的映像标签 | <code>2024.6.3</code> |
| <code>binaryscanner.resources.limits.Cpu</code> | 二进制扫描程序容器 CPU 限制 | 1000m |
| <code>binaryscanner.resources.requests.Cpu</code> | 二进制扫描程序容器 CPU 请求 | 1000m |
| <code>binaryscanner.resources.limits.memory</code> | 二进制扫描程序容器内存限制 | 2048Mi |
| <code>binaryscanner.resources.requests.memory</code> | 二进制扫描程序容器内存请求 | 2048Mi |
| <code>binaryscanner.nodeSelector</code> | 用于 Pod 分配的二进制扫描程序节点标签 | <code>{}</code> |
| <code>binaryscanner.tolerations</code> | 用于 Pod 分配的二进制扫描程序节点容忍度 | <code>[]</code> |

| 参数 | 说明 | 默认 |
|----------------------------------|------------------------|----|
| binaryscanner.affinity | 用于 Pod 分配的二进制扫描程序节点亲和性 | {} |
| binaryscanner.podSecurityContext | Pod 级别的二进制扫描程序安全上下文 | {} |
| binaryscanner.securityContext | 容器级别的二进制扫描程序安全上下文 | {} |

BOM 引擎 Pod 配置

| 参数 | 说明 | 默认 |
|-------------------------------------|------------------------|--------|
| bomengine.registry | 要在容器级别覆盖的映像存储库 | |
| bomengine.resources.limits.memory | BOM 引擎容器内存限制 | 1024Mi |
| bomengine.resources.requests.memory | BOM 引擎容器内存请求 | 1024Mi |
| bomengine.maxRamPercentage | BOM 引擎最大堆大小 | 90 |
| bomengine.nodeSelector | 用于 Pod 分配的 BOM 引擎节点标签 | {} |
| bomengine.tolerations | 用于 Pod 分配的 BOM 引擎节点容忍度 | [] |
| bomengine.affinity | 用于 Pod 分配的 BOM 引擎节点亲和性 | {} |
| bomengine.podSecurityContext | Pod 级别的 BOM 引擎安全上下文 | {} |
| bomengine.securityContext | 容器级别的 BOM 引擎安全上下文 | {} |

CFSSL Pod 配置

| 参数 | 说明 | 默认 |
|---------------------------------|-------------------------|--------|
| cfssl.registry | 要在容器级别覆盖的映像存储库 | |
| cfssl.imageTag | 要在容器级别覆盖的映像标签 | 1.0.28 |
| cfssl.resources.limits.memory | Cfssl 容器内存限制 | 640Mi |
| cfssl.resources.requests.memory | Cfssl 容器内存请求 | 640Mi |
| cfssl.persistentVolumeClaimName | 指向现有的 Cfssl 持久卷声明 (PVC) | |
| cfssl.claimSize | Cfssl 持久卷声明 (PVC) 声明大小 | 2Gi |
| cfssl.storageClass | Cfssl 持久卷声明 (PVC) 存储类 | |
| cfssl.volumeName | 指向现有的 Cfssl 持久卷 (PV) | |
| cfssl.nodeSelector | 用于 Pod 分配的 Cfssl 节点标签 | {} |
| cfssl.tolerations | 用于 Pod 分配的 Cfssl 节点容忍度 | [] |
| cfssl.affinity | 用于 Pod 分配的 Cfssl 节点亲和性 | {} |

| 参数 | 说明 | 默认 |
|--------------------------|---------------------|----|
| cfssl.podSecurityContext | Pod 级别的 Cfssl 安全上下文 | {} |
| cfssl.securityContext | 容器级别的 Cfssl 安全上下文 | {} |

Datadog Pod 配置

| 参数 | 说明 | 默认 |
|-------------------------|-----------------------------------------------------|--------------|
| datadog.enable | 仅对托管客户为 true (Values.enableInitContainer 应为 true) | false |
| datadog.registry | 要在容器级别覆盖的映像存储库 | |
| datadog.imageTag | 要在容器级别覆盖的映像标签 | 1.0.15 |
| datadog.imagePullPolicy | 映像拉取策略 | IfNotPresent |

文档 Pod 配置

| 参数 | 说明 | 默认 |
|-----------------------------------------|-------------------|-------|
| documentation.registry | 要在容器级别覆盖的映像存储库 | |
| documentation.resources.limits.memory | 文档容器内存限制 | 512Mi |
| documentation.resources.requests.memory | 文档容器内存请求 | 512Mi |
| documentation.maxRamPercentage | 文档容器内存请求 | 90 |
| documentation.nodeSelector | 用于 Pod 分配的文档节点标签 | {} |
| documentation.tolerations | 用于 Pod 分配的文档节点容忍度 | [] |
| documentation.affinity | 用于 Pod 分配的文档节点亲和性 | {} |
| documentation.podSecurityContext | Pod 级别的文档安全上下文 | {} |
| documentation.securityContext | 容器级别的文档安全上下文 | {} |

集成 Pod 配置

| 参数 | 说明 | 默认 |
|---------------------------------------|------------------|--------|
| integration.registry | 要在容器级别覆盖的映像存储库 | |
| integration.replicas | 集成 Pod 副本计数 | 1 |
| integration.resources.limits.cpu | 集成容器 CPU 限制 | 1000m |
| integration.resources.requests.cpu | 集成容器 CPU 请求 | 500m |
| integration.resources.limits.memory | 集成容器内存限制 | 5120Mi |
| integration.resources.requests.memory | 集成容器内存请求 | 5120Mi |
| integration.maxRamPercentage | 集成容器最大堆大小 | 90 |
| integration.nodeSelector | 用于 Pod 分配的集成节点标签 | {} |

| 参数 | 说明 | 默认 |
|--------------------------------|-------------------|----|
| integration.tolerations | 用于 Pod 分配的集成节点容忍度 | [] |
| integration.affinity | 用于 Pod 分配的集成节点亲和性 | {} |
| integration.podSecurityContext | Pod 级别的集成安全上下文 | {} |
| integration.securityContext | 容器级别的集成安全上下文 | {} |

作业运行器 Pod 配置

| 参数 | 说明 | 默认 |
|-------------------------------------|----------------------|--------|
| jobrunner.registry | 要在容器级别覆盖的映像存储库 | |
| jobrunner.replicas | 作业运行器 Pod 副本计数 | 1 |
| jobrunner.resources.limits.cpu | 作业运行器容器 CPU 限制 | 1000m |
| jobrunner.resources.requests.cpu | 作业运行器容器 CPU 请求 | 1000m |
| jobrunner.resources.limits.memory | 作业运行器容器内存限制 | 4608Mi |
| jobrunner.resources.requests.memory | 作业运行器容器内存请求 | 4608Mi |
| jobrunner.maxRamPercentage | 作业运行器容器最大堆大小 | 90 |
| jobrunner.nodeSelector | 用于 Pod 分配的作业运行器节点标签 | {} |
| jobrunner.tolerations | 用于 Pod 分配的作业运行器节点容忍度 | [] |
| jobrunner.affinity | 用于 Pod 分配的作业运行器节点亲和性 | {} |
| jobrunner.podSecurityContext | Pod 级别的作业运行器安全上下文 | {} |
| jobrunner.securityContext | 容器级别的作业运行器安全上下文 | {} |

Logstash Pod 配置

| 参数 | 说明 | 默认 |
|------------------------------------|----------------------------|--------|
| logstash.registry | 要在容器级别覆盖的映像存储库 | |
| logstash.imageTag | 要在容器级别覆盖的映像标签 | 1.0.38 |
| logstash.resources.limits.memory | Logstash 容器内存限制 | 1024Mi |
| logstash.resources.requests.memory | Logstash 容器内存请求 | 1024Mi |
| logstash.maxRamPercentage | Logstash 最大堆大小 | 90 |
| logstash.persistentVolumeClaimName | 指向现有的 Logstash 持久卷声明 (PVC) | |
| logstash.claimSize | Logstash 持久卷声明 (PVC) 声明大小 | 20Gi |

4. 使用 Helm 安装 Black Duck • 配置参数

| 参数 | 说明 | 默认 |
|--------------------------|---------------------------|----|
| logstash.storageClass | Logstash 持久卷声明 (PVC) 存储类 | |
| logstash.volumeName | 指向现有的 Logstash 持久卷 (PV) | |
| logstash.nodeSelector | 用于 Pod 分配的 Logstash 节点标签 | {} |
| logstash.tolerations | 用于 Pod 分配的 Logstash 节点容忍度 | [] |
| logstash.affinity | 用于 Pod 分配的 Logstash 节点亲和性 | {} |
| logstash.securityContext | 容器级别的 Logstash 安全上下文 | {} |

匹配引擎 Pod 配置

| 参数 | 说明 | 默认 |
|---------------------------------------|---------------------|--------|
| matchengine.registry | 要在容器级别覆盖的映像存储库 | |
| matchengine.resources.limits.memory | 匹配引擎容器内存限制 | 4608Mi |
| matchengine.resources.requests.memory | 匹配引擎容器内存请求 | 4608Mi |
| matchengine.maxRamPercentage | 匹配引擎最大堆大小 | 90 |
| matchengine.nodeSelector | 用于 Pod 分配的匹配引擎节点标签 | {} |
| matchengine.tolerations | 用于 Pod 分配的匹配引擎节点容忍度 | [] |
| matchengine.affinity | 用于 Pod 分配的匹配引擎节点亲和性 | {} |
| matchengine.podSecurityContext | Pod 级别的匹配引擎安全上下文 | {} |
| matchengine.securityContext | 容器级别的匹配引擎安全上下文 | {} |

PostgreSQL Pod 配置

| 参数 | 说明 | 默认 |
|--------------------------------|---------------------------------------|------------------------------|
| postgres.registry | 映像存储库 | docker.io/centos |
| postgres.isExternal | 如果为 true，将使用外部 PostgreSQL | true |
| postgres.host | PostgreSQL 主机（仅当使用外部 PostgreSQL 时才需要） | |
| postgres.port | PostgreSQL 端口 | 5432 |
| postgres.pathToPgsqlInitScript | PostgreSQL 初始化脚本的完整文件路径 | external-postgres-init.pgsql |
| postgres.ssl | PostgreSQL SSL | true |

| 参数 | 说明 | 默认 |
|----------------------------------------|-----------------------------------------------------|----------------|
| postgres.adminUserName | PostgreSQL 管理员用户名 | postgres |
| postgres.adminPassword | PostgreSQL 管理员用户密码 | testPassword |
| postgres.userUserName | PostgreSQL 非管理员用户名 | blackduck_user |
| postgres.userPassword | PostgreSQL 非管理员用户密码 | testPassword |
| postgres.resources.requests.cpu | PostgreSQL 容器 CPU 请求 (如果未使用外部 postgres) | 1000m |
| postgres.resources.requests.memory | PostgreSQL 容器内存请求 (如果未使用外部 postgres) | 3072Mi |
| postgres.persistentVolumeClaimName | 指向现有的 PostgreSQL 持久卷声明 (PVC) (如果未使用外部 postgres) | |
| postgres.claimSize | PostgreSQL 持久卷声明 (PVC) 声明大小 (如果未使用外部 postgres) | 150Gi |
| postgres.storageClass | PostgreSQL 持久卷声明 (PVC) 存储类 (如果未使用外部 postgres) | |
| postgres.volumeName | 指向现有的 PostgreSQL 持久卷 (PV) (如果未使用外部 postgres) | |
| postgres.confPersistentVolumeClaimName | 指向现有的 PostgreSQL 配置持久卷声明 (PVC) (如果未使用外部 postgres) | |
| postgres.confClaimSize | PostgreSQL 配置持久卷声明 (PVC) 声明大小 (如果未使用外部 postgres) | 5Mi |
| postgres.confStorageClass | PostgreSQL 配置持久卷声明 (PVC) 存储类 (如果未使用外部 postgres) | |
| postgres.confVolumeName | 指向现有的 PostgreSQL 配置持久卷 (PV) (如果未使用外部 postgres) | |
| postgres.nodeSelector | 用于 Pod 分配的 PostgreSQL 节点 标签 | {} |
| postgres.tolerations | 用于 Pod 分配的 PostgreSQL 节点 容忍度 | [] |
| postgres.affinity | 用于 Pod 分配的 PostgreSQL 节点 亲和性 | {} |
| postgres.podSecurityContext | Pod 级别的 PostgreSQL 安全上下文 | {} |
| postgres.securityContext | 容器级别的 PostgreSQL 安全上下文 | {} |

PostgreSQL 就绪性初始化容器配置

| 参数 | 说明 | 默认 |
|-----------------------------------|----------------------------|----|
| postgresWaiter.registry | 映像存储库 | |
| postgresWaiter.podSecurityContext | Pod 级别的 Postgre 就绪性检查安全上下文 | {} |
| postgresWaiter.securityContext | 容器级别的 Postgre 就绪性检查安全上下文 | {} |

PostgreSQL 升级作业配置

| 参数 | 说明 | 默认 |
|-------------------------------------|-------------------------|----|
| postgresUpgrader.registry | 映像存储库 | |
| postgresUpgrader.podSecurityContext | 作业级别的 Postgres 升级器安全上下文 | {} |
| postgresUpgrader.securityContext | 容器级别的 Postgres 升级器安全上下文 | {} |

RabbitMQ Pod 配置

| 参数 | 说明 | 默认 |
|------------------------------------|---------------------------|--------|
| rabbitmq.registry | 要在容器级别覆盖的映像存储库 | |
| rabbitmq.imageTag | 要在容器级别覆盖的映像标签 | 1.2.40 |
| rabbitmq.resources.limits.memory | RabbitMQ 容器内存限制 | 1024Mi |
| rabbitmq.resources.requests.memory | RabbitMQ 容器内存请求 | 1024Mi |
| rabbitmq.nodeSelector | 用于 Pod 分配的 RabbitMQ 节点标签 | {} |
| rabbitmq.tolerations | 用于 Pod 分配的 RabbitMQ 节点容忍度 | [] |
| rabbitmq.affinity | 用于 Pod 分配的 RabbitMQ 节点亲和性 | {} |
| rabbitmq.podSecurityContext | Pod 级别的 RabbitMQ 安全上下文 | {} |
| rabbitmq.securityContext | 容器级别的 RabbitMQ 安全上下文 | {} |

Redis Pod 配置

| 参数 | 说明 | 默认 |
|---------------------------------|----------------|--------|
| redis.registry | 要在容器级别覆盖的映像存储库 | |
| redis.resources.limits.memory | Redis 容器内存限制 | 1024Mi |
| redis.resources.requests.memory | Redis 容器内存请求 | 1024Mi |

| 参数 | 说明 | 默认 |
|--------------------------|---------------------------------|-------|
| redis.tlsEnalbed | 启用客户端和 Redis 之间的 TLS 连接 | false |
| redis.maxTotal | 可连接到 Redis 的最大并发客户端连接数 | 128 |
| redis.maxIdle | 在不释放额外连接的情况下，池中可保持空闲的最大并发客户端连接数 | 128 |
| redis.nodeSelector | 用于 Pod 分配的 Redis 节点标签 | {} |
| redis.tolerations | 用于 Pod 分配的 Redis 节点容忍度 | [] |
| redis.affinity | 用于 Pod 分配的 Redis 节点亲和性 | {} |
| redis.podSecurityContext | Pod 级别的 Redis 安全上下文 | {} |
| redis.securityContext | 容器级别的 Redis 安全上下文 | {} |

注册 Pod 配置

| 参数 | 说明 | 默认 |
|----------------------------------------|--------------------|--------|
| registration.registry | 要在容器级别覆盖的映像存储库 | |
| registration.requestCpu | 注册容器 CPU 请求 | 1000m |
| registration.resources.limits.memory | 注册容器内存限制 | 1024Mi |
| registration.resources.requests.memory | 注册容器内存请求 | 1024Mi |
| registration.maxRamPercentage | 注册容器最大堆大小 | 90 |
| registration.persistentVolumeClaimName | 指向现有的注册持久卷声明 (PVC) | |
| registration.claimSize | 注册持久卷声明 (PVC) 声明大小 | 2Gi |
| registration.storageClass | 注册持久卷声明 (PVC) 存储类 | |
| registration.volumeName | 指向现有的注册持久卷 (PV) | |
| registration.nodeSelector | 用于 Pod 分配的注册节点标签 | {} |
| registration.tolerations | 用于 Pod 分配的注册节点容忍度 | [] |
| registration.affinity | 用于 Pod 分配的注册节点亲和性 | {} |
| registration.podSecurityContext | Pod 级别的注册安全上下文 | {} |
| registration.securityContext | 容器级别注册安全上下文 | {} |

扫描 pod 配置

| 参数 | 说明 | 默认 |
|---------------|----------------|----|
| scan.registry | 要在容器级别覆盖的映像存储库 | |

| 参数 | 说明 | 默认 |
|--------------------------------|-------------------|--------|
| scan.replicas | 扫描 Pod 副本计数 | 1 |
| scan.resources.limits.memory | 扫描容器内存限制 | 2560Mi |
| scan.resources.requests.memory | 扫描容器内存请求 | 2560Mi |
| scan.maxRamPercentage | 扫描容器最大堆大小 | 90 |
| scan.nodeSelector | 用于 Pod 分配的扫描节点标签 | {} |
| scan.tolerations | 用于 Pod 分配的扫描节点容忍度 | [] |
| scan.affinity | 用于 Pod 分配的扫描节点亲和性 | {} |
| scan.podSecurityContext | Pod 级别的扫描安全上下文 | {} |
| scan.securityContext | 容器级别的扫描安全上下文 | {} |

存储 Pod 配置

| 参数 | 说明 | 默认 |
|-----------------------------------|---------------------------------|--------|
| storage.registry | 要在容器级别覆盖的映像存储库 | |
| storage.requestCpu | 存储容器 CPU 请求 | 1000m |
| storage.resources.limits.memory | 存储容器内存限制 | 2048Mi |
| storage.resources.requests.memory | 存储容器内存请求 | 2048Mi |
| storage.maxRamPercentage | 存储容器最大堆大小 | 60 |
| storage.persistentVolumeClaimName | 指向现有的存储持久卷声明 (PVC) | |
| storage.claimSize | 存储持久卷声明 (PVC) 声明大小 | 100Gi |
| storage.storageClass | 存储持久卷声明 (PVC) 存储类 | |
| storage.volumeName | 指向现有的存储持久卷 (PV) | |
| storage.nodeSelector | 用于 Pod 分配的存储节点标签 | {} |
| storage.tolerations | 用于 Pod 分配的存储节点容忍度 | [] |
| storage.affinity | 用于 Pod 分配的存储节点亲和性 | {} |
| storage.podSecurityContext | Pod 级别的存储安全上下文 | {} |
| storage.securityContext | 容器级别的存储安全上下文 | {} |
| storage.providers | 支持多种存储平台的配置。请参阅存储提供者部分了解更多详细信息。 | [] |

存储提供者

存储服务中的提供者指的是持久化类型及其配置。Black Duck 管理存储服务下的工具、应用程序报告和其他大型 blob。目前，它仅支持文件系统作为提供者之一。

```
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>

```

| 参数 | 类型 | 说明 | 默认 |
|-----------------------------------|---------|------------------------------------------------------|--------|
| name | String | 提供者配置的名称。例如 blackduck-file-storage | |
| enabled | Boolean | 用于控制启用/禁用提供者实例的标志 | false |
| index | Integer | 用于提供者配置的索引值。例如1、2、3。 | |
| type | String | 存储类型。默认为 file。 | file |
| preference | Integer | 表示提供者实例配置权重的数字。如果配置了多个提供者实例，则该值用于确定将哪个提供者用作默认存储选项。 | |
| existingPersistentVolumeClaimName | String | 为提供者重新使用现有持久卷声明的选项 | |
| pvc.size | String | 创建持久卷时要使用的卷大小。建议存储服务的最小大小为 100Gi。 | 100Gi |
| pvc.storageClass | String | 用于持久卷的存储类 | |
| pvc.existingPersistentVolumeName | String | 为提供者重新使用现有持久卷的选项 | |
| mountPath | String | 要挂载提供者卷的容器内路径 | |
| readonly | Boolean | 如果存在，则可将提供者标记为只读 | false |
| migrationMode | String | 表示是否配置了迁移。值可以是 "NONE"、"DRAIN"、"DELETE" 或 "DUPLICATE" | "NONE" |

Webapp Pod 配置

| 参数 | 说明 | 默认 |
|--------------------------------|------------------|--------|
| webapp.registry | 要在容器级别覆盖的映像存储库 | |
| webapp.resources.requests.cpu | Webapp 容器 CPU 请求 | 1000m |
| webapp.resources.limits.memory | Webapp 容器内存限制 | 2560Mi |

4. 使用 Helm 安装 Black Duck • 配置参数

| 参数 | 说明 | 默认 |
|----------------------------------|---------------------------------|--------|
| webapp.resources.requests.memory | Webapp 容器内存请求 | 2560Mi |
| webapp.maxRamPercentage | Webapp 容器最大堆大小 | 90 |
| webapp.persistentVolumeClaimName | 指向现有的 Webapp 持久卷声明 (PVC) | |
| webapp.claimSize | Webapp 持久卷声明 (PVC) 声明大小 | 2Gi |
| webapp.storageClass | Webapp 持久卷声明 (PVC) 存储类 | |
| webapp.volumeName | 指向现有的 Webapp 持久卷 (PV) | |
| webapp.nodeSelector | 用于 Pod 分配的 Webapp 节点标签 | {} |
| webapp.tolerations | 用于 Pod 分配的 Webapp 节点容忍度 | [] |
| webapp.affinity | 用于 pod 分配的 Webapp 节点亲和性 | {} |
| webapp.podSecurityContext | Pod 级别的 Webapp 和 Logstash 安全上下文 | {} |
| webapp.securityContext | 容器级别的 Webapp 安全上下文 | {} |

Webserver Pod 配置

| 参数 | 说明 | 默认 |
|-------------------------------------|----------------------------|----------|
| webserver.registry | 要在容器级别覆盖的映像存储库 | |
| webserver.imageTag | 要在容器级别覆盖的映像标签 | 2024.7.1 |
| webserver.resources.limits.memory | Webserver 容器内存限制 | 512Mi |
| webserver.resources.requests.memory | Webserver 容器内存请求 | 512Mi |
| webserver.nodeSelector | 用于 Pod 分配的 Webserver 节点标签 | {} |
| webserver.tolerations | 用于 Pod 分配的 Webserver 节点容忍度 | [] |
| webserver.affinity | 用于 Pod 分配的 Webserver 节点亲和性 | {} |
| webserver.podSecurityContext | Pod 级别的 Webserver 安全上下文 | {} |
| webserver.securityContext | 容器级别的 Webserver 安全上下文 | {} |


5. 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（root 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
```

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，可以在 Kubernetes 中创建和删除密钥。

```
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。

根密钥用于包装实际加密和解密 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。

通过 Kubernetes 密钥传递外部数据库凭据

在配置 Black Duck 以使用外部 PostgreSQL 数据库时，您可以选择通过 Kubernetes 密钥提供数据库凭据，而不是直接将凭据存储在 `values.yaml` 文件中。这种方法避免了在配置文件中存储明文凭据，从而增强了安全性。

使用默认行为（Helm 管理的密钥）

默认情况下，Helm 图表将使用 `values.yaml` 文件中 `adminPassword` 和 `userPassword` 设置的值生成名为 `<name>-blackduck-db-creds` 的密钥。此行为由默认启用的 `useHelmChartDbCreds` 标志控制：

```
useHelmChartDbCreds: true
```

如果选择继续使用这种方法，则无需其他步骤。

提供自己的数据库凭据密钥

如果您更希望自己管理凭据，请在 values.yaml 文件中将 useHelmChartDbCreds 设置为 false：

```
useHelmChartDbCreds: false
```

然后，您必须在与 Black Duck 部署相同的命名空间中创建名为 <name>-blackduck-db-creds 的 Kubernetes 密钥。该密钥必须包括以下键：

- HUB_POSTGRES_ADMIN_PASSWORD_FILE
- HUB_POSTGRES_USER_PASSWORD_FILE

每个键都应指向包含相应密码的文件。例如：

```
kubectrl 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：如果自定义密钥无效或丢失，部署将失败。Helm 不会退回到使用 values.yaml 中指定的凭据。

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。配置文件中的顺序并不重要。 |
| type | 默认值：file。 有效值：file。 备注："file" 是唯一受支持的提供商类型。 |
| preference | 默认值：index 乘以 10。 有效值：0-999。 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。 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。 |
| readonly | 默认值：false。 有效值：true 或 false。 备注：表示提供商为只读状态。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。 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。 |
| migrationMode | 默认值：none。 有效值：none、drain、delete、duplicate。 |

| 设置 | 详细信息 |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | 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。 |
| existingPersistentVolumeClaimName | 默认值：""。 有效值：任何有效的 k8s 标识符。 备注：允许您为该卷指定特定的持久性卷声明名称。 |
| pvc.size | 默认值：none。 有效值：任何有效大小。 备注：允许您指定该卷的可用空间量。 |
| pvc.storageClass | 默认值：""。 有效值：任何有效的 k8s 标识符。 备注：允许您为该卷指定特定的存储类。 |
| pvc.existingPersistentVolumeName | 默认值：""。 有效值：任何有效的 k8s 标识符。 备注：允许您为该卷指定特定的持久性卷名称。 |
| mountPath | 默认值：特定于索引 - 请参阅注释。 有效值： /opt/blackduck/hub/uploads /opt/blackduck/hub/uploads2 /opt/blackduck/hub/uploads3 备注： 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 /opt/blackduck/hub/uploads。索引为二 (2) 的提供商必须指定挂载点 /opt/blackduck/hub/uploads2。索引为三 (3) 的提供商必须指定挂载点 /opt/blackduck/hub/uploads3 |

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。 |

| 迁移模式 | 详细信息 |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | 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
```

配置就绪探针

通过编辑以下 values.yaml 中的布尔值标志，您可以启用或禁用就绪探针：

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

配置 HUB_MAX_MEMORY 设置

在基于 Kubernetes 的部署中，会自动为相关容器设置配置参数 HUB_MAX_MEMORY。该值按内存限制的百分比计算，默认值为 90%。

在 gen04 部署调整中，maxRamPercentage 控制使用的百分比；选择此设置的值，以使 HUB_MAX_MEMORY 具有与之前相同的值。

使用 Helm 在 OpenShift 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 Black Duck 提供的容器替换 CentOS PostgreSQL 容器。此外，blackduck-init 容器会被替换为 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。

预置 JWT 公钥/私钥对

为了增强 JWT 管理的安全性和灵活性，我们的系统现在支持可选的公钥/私钥对预置。这使您可以安全地提供和管理这些密钥，确保它们仅由适当的服务使用，如身份验证服务使用私钥，公共 API 服务使用公钥。

目前，仅支持 RSA 密钥（PEM 编码）。具体来说，公钥必须采用 X.509 格式，私钥必须采用 PKCS#8 格式。

创建 Kubernetes 密钥

1. 创建 Kubernetes 密钥（模板命令）。必须为公钥和私钥选项提供准确的文件。

```
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
```

如果 namespace = bd 并且 name = hub，以下是一个示例命令：

```
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. 确认在 namespace 中创建的密钥：

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

如果 namespace = bd 并且密钥名称 = hub-blackduck-jwt-keypair，预期输出如下所示：

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

3. 取消注释 values.yaml 中的以下一行，并根据密钥名称相应地编辑 name：

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


4. 在相同的 namespace 中部署 Black Duck。例如：

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

启用 SCM 集成

Black Duck 默认不启用此功能，必须将此功能添加到[产品注册密钥](#)中，然后在 values.yaml 文件中添加以下内容才能激活：

```
enableIntegration: true
```

 注：Black Duck 目前不接受 SCM 集成的自签名证书。

在外部数据库中配置 mTLS

前提条件

在 Kubernetes 部署中为外部数据库配置 mTLS 之前，请确保满足以下条件：

1. 环境设置：
 - 已安装并适当配置 Kubernetes 和 Helm。
 - 已在环境中设置持久卷 (PV) 和持久卷声明 (PVC)。
2. 部署准备：
 - 您已确定要使用的正确部署文件并知道在哪里下载这些文件。
 - 您熟悉使用外部数据库设置 Black Duck 部署。
 - 您已准备好使用外部数据库运行 Black Duck 部署所需的命令。
3. mTLS 要求：
 - 您的服务器上提供了用于生成证书的 openssl.cnf 文件。
 - 已使用部署说明中提供的默认值对密钥进行命名。
 - 根据您的部署，您可能有最多五个 mTLS 密钥：根证书、管理员证书和密钥以及用户证书和密钥。如果您没有使用这些密钥中的一个或多个，请将 values.yaml 文件中的相应条目更新为空字符串 ("")。
 - 示例：要配置根证书，请更新 value.yaml 中的 postgres.customCerts.rootCAKeyName 字段。默认情况下，此字段设置为 "HUB_POSTGRES_CA"。如果您没有根证书，请将此值设置为 ""。
4. 命名空间一致性：
 - Black Duck 和 PostgreSQL 部署使用相同的命名空间。

Black Duck 部署的更改

更新 values.yaml

要为您的 Black Duck 部署配置 mTLS，则必须使用必要的设置密钥来更新 values.yaml 文件。按照以下步骤操作：

1. 配置 postgres.sslMode

将 postgres.sslMode 配置选项添加到 values.yaml。此选项确定部署是否包含证书和密钥。如果未配置此选项，setenv.sh 脚本将从部署中排除证书/密钥。确保将此值设置为启用 mTLS。
2. 设置自定义证书的密钥名称

将 postgres.customCerts.useCustomCerts 选项添加到 values.yaml。此选项指定包含连接到外部数据库所需证书和密钥数据的密钥（例如，根 CA、管理员证书/密钥和用户证书/密钥）的名称。
3. 定义自定义证书数据

在 values.yaml 中的 postgres.customCerts 部分下面，配置以下五个选项来指定证书和密钥的数据：

 - rootCAKeyName：根 CA 证书的密钥名称。
 - clientCertName：用户证书的密钥名称。

- clientKeyName：用户私钥的密钥名称。
- adminClientCertName：管理员证书的密钥名称。
- adminClientKeyName：管理员私钥的密钥名称。

确保这些值与您的密钥配置中的名称匹配。

更新 postgres-init.yaml

为了支持 mTLS，需要在 postgres-init.yaml 文件中进行以下更改：

1. 为五个新的 SSL 密钥添加卷和卷挂载逻辑。
2. 包括用于设置 PGSSLMODE 环境变量的逻辑。
3. 添加逻辑来设置 PGSSLROOTCERT 环境变量。
4. 先根据 HUB_POSTGRES_CERT 设置 PGSSLCERT 环境变量，然后根据 HUB_ADMIN_POSTGRES_CERT 进行设置。为了清楚起见，应使用单独的 if 语句。
5. 先根据 HUB_POSTGRES_KEY 设置 PGSSLKEY 环境变量，然后根据 HUB_ADMIN_POSTGRES_KEY 进行设置。为了清楚起见，应使用单独的 if 语句。
6. 通过移动用于运行 /tmp/postgres-init/init.pgsql 的逻辑并将其与其他 shell 命令分组在一起，对 shell 命令逻辑进行重新组织，以提高可读性。

更新环境

将 HUB_POSTGRES_ENABLE_SSL_CERT_AUTH: "true" 添加到环境中。

更新 postgres.name

启动数据库 Pod 后，检索 Pod 的主机值并在 values.yaml 中更新 postgres.host 字段。

更新 postgres.customCerts.useCustomCerts

将 postgres.customCerts.useCustomCerts 更新为 true。

为证书和密钥创建密钥

使用以下命令为证书和密钥创建密钥：

```
kubectl 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
```

从 Synopsysctl 转换到 Helm 图表部署

随着 Black Duck 的演变，我们正在从使用 synopsysctl 转换到使用 Helm 图表来管理 Kubernetes 部署。Helm 图表提供了一种更加标准化和灵活的方法，在 Kubernetes 环境中部署、升级和维护 Black Duck。

在本指南中，我们建议全新安装，因为升级现有部署可能会因卷命名约定和其他配置差异而带来风险。

！ 重要： 在开始转换过程之前，请确保备份数据库和任何其他关键数据。此步骤至关重要，可以防止在转换期间因出现意外问题而丢失数据。在继续之前，请务必验证备份的完整性。


我们强烈建议先在测试环境中部署 Black Duck，然后再部署到生产环境。这样您就可以验证流程并识别潜在问题。测试环境可以是专用测试实例，也可以是从生产环境克隆的临时实例。

内部或外部数据库

如果使用外部数据库，您可以将新安装配置为连接到现有数据库，前提是在任何给定时间只有一个 Black Duck 实例与其通信。或者，您可以创建一个新数据库并对现有数据执行备份和恢复。

如果使用内部数据库，则必须备份当前数据库并将其恢复到新实例。

转换过程

1. 从生产环境备份数据库。
2. 使用 Helm 在新命名空间中部署全新安装的 Black Duck。
 注：如果使用外部数据库，请在部署期间将新安装配置为指向现有外部数据库。
3. 验证 Black Duck 实例是否正常运行。
4. 使用生产备份恢复数据库（对于内部或外部数据库）。
5. 执行彻底的测试以确保功能性和数据完整性。
6. 如果执行模拟运行并且成功，请在停止生产实例并更新 DNS 路由或负载均衡器以指向新实例后重复上述步骤。