OpenShift Container Platform 4.10

CI/CD

包含有关 OpenShift Container Platform 构建、管道和 GitOps 的信息

Last Updated: 2023-10-17

OpenShift Container Platform 4.10 CI/CD

包含有关 OpenShift Container Platform 构建、管道和 GitOps 的信息

法律通告

摘要

OpenShift Container Platform 的 CI/CD

目录

第 1 章 OPENSHIFT CONTAINER PLATFORM CI/CD 概述

OpenShift Container Platform 是面向开发人员的企业就绪 Kubernetes 平台，使组织能够通过 DevOps 实践（如持续集成(CI)和持续交付(CD)）自动化应用程序交付流程。为了满足您的机构需求，OpenShift Container Platform 提供以下 CI/CD 解决方案：

- OpenShift 构建
- OpenShift Pipelines
- OpenShift GitOps

1.1. OPENSHIFT 构建

使用 OpenShift 构建时，您可以使用声明性构建过程创建云原生应用程序。您可以在用于创建 BuildConfig 对象的 YAML 文件中定义构建过程。此定义包括构建触发器、输入参数和源代码等属性。部署之后，BuildConfig 对象通常构建可运行的镜像并将其推送到容器镜像 registry。

OpenShift 构建为构建策略提供以下可扩展的支持：

- Docker 构建
- Source-to-image（S2I）构建
- Custom 构建

如需更多信息，请参阅了解镜像构建

1.2. OPENSHIFT PIPELINES

OpenShift Pipelines 提供了一个 Kubernetes 原生 CI/CD 框架，用于在其自己的容器中设计和运行 CI/CD 管道的每个步骤。它可以通过可预测的结果独立扩展以满足按需管道。

如需更多信息，请参阅了解 OpenShift Pipelines

1.3. OPENSHIFT GITOPS

OpenShift GitOps 是一个使用 Argo CD 作为声明性 GitOps 引擎的 Operator。它启用了多集群 OpenShift 和 Kubernetes 基础架构的 GitOps 工作流。使用 OpenShift GitOps，管理员可以在集群和开发生命周期中一致地配置和部署基于 Kubernetes 的基础架构和应用程序。

如需更多信息，请参阅了解 OpenShift GitOps

1.4. JENKINS

Jenkins 自动化了构建、测试和部署应用和项目的过程。OpenShift 开发者工具提供 Jenkins 镜像，它直接与 OpenShift Container Platform 集成。Jenkins 可通过使用 Samples Operator 模板或认证的 Helm Chart 在 OpenShift 上部署。

第 2 章 构建（BUILD）

2.1. 理解镜像构建

2.1.1. Builds

构建 (build) 是将输入参数转换为结果对象的过程。此过程最常用于将输入参数或源代码转换为可运行的镜像。**BuildConfig** 对象是整个构建过程的定义。

OpenShift Container Platform 使用 Kubernetes，从构建镜像创建容器并将它们推送到容器镜像 registry。

构建对象具有共同的特征，包括构建的输入，完成构建过程要满足的要求、构建过程日志记录、从成功构建中发布资源，以及发布构建的最终状态。构建会使用资源限制，具体是指定资源限值，如 CPU 使用量、内存使用量，以及构建或 Pod 执行时间。

OpenShift Container Platform 构建系统提供对构建策略的可扩展支持，它们基于构建 API 中指定的可选择类型。可用的构建策略主要有三种：

- Docker 构建
- Source-to-image（S2I）构建
- Custom 构建

默认情况下，支持 docker 构建和 S2I 构建。

构建生成的对象取决于用于创建它的构建器（builder）。对于 docker 和 S2I 构建，生成的对象为可运行的镜像。对于自定义构建，生成的对象是构建器镜像作者指定的任何事物。

此外，也可利用管道构建策略来实现复杂的工作流：

- 持续集成
- 持续部署

2.1.1.1. Docker 构建

OpenShift Container Platform 使用 Buildah 从 Dockerfile 构建容器镜像。有关使用 Dockerfile 构建容器镜像的更多信息，请参阅 Dockerfile 参考文档。

提示

如果使用 **buildArgs** 数组设置 Docker 构建参数，请参阅 Dockerfile 参考文档中 了解 ARG 和 FROM 如何交互。

2.1.1.2. Source-to-image 构建

Source-to-Image (S2I) 是一种用于构建可重复生成的容器镜像的工具。它通过将应用程序源代码注入容器镜像并汇编新镜像来生成可随时运行的镜像。新镜像融合了基础镜像（构建器）和构建的源代码，并可搭配 **buildah run** 命令使用。S2I 支持递增构建，可重复利用以前下载的依赖项和过去构建的工件等。

2.1.1.3. Custom 构建

采用自定义构建策略时，开发人员可以定义负责整个构建过程的特定构建器镜像。通过利用自己的构建器镜像，可以自定义构建流程。

自定义构建器镜像是嵌入了构建过程逻辑的普通容器镜像，例如用于构建 RPM 或基本镜像的逻辑。

自定义构建以级别很高的特权运行，默认情况下不可供用户使用。只有可赋予集群管理权限的用户才应被授予运行自定义构建的权限。

2.1.1.4. Pipeline 构建

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

采用 Pipeline 构建策略时，开发人员可以定义 Jenkins 管道，供 Jenkins 管道插件使用。构建可以由 OpenShift Container Platform 启动、监控和管理，其方式与任何其他构建类型相同。

Pipeline 工作流在 **jenkinsfile** 中定义，或直接嵌入在构建配置中，或者在 Git 存储库中提供并由构建配置引用。

2.2. 了解构建配置

以下小节定义了构建、构建配置和可用的主要构建策略的概念。

2.2.1. BuildConfig

构建配置描述单个构建定义，以及一组规定何时创建新构建的触发器（trigger）。构建配置通过 **BuildConfig** 定义，它是一种 REST 对象，可在对 API 服务器的 POST 中使用以创建新实例。

构建配置或 **BuildConfig** 的特征就是构建策略和一个或多个源。策略决定流程，而源则提供输入。

根据您选择使用 OpenShift Container Platform 创建应用程序的方式，如果使用 Web 控制台或 CLI，通常会自动生成 **BuildConfig**，并且可以随时对其进行编辑。如果选择稍后手动更改配置，则了解 **BuildConfig** 的组成部分及可用选项可能会有所帮助。

以下示例 **BuildConfig** 在每次容器镜像标签或源代码改变时产生新的构建：

BuildConfig 对象定义

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" 1
spec:
  runPolicy: "Serial" 2
  triggers: 3
    -
      type: "GitHub"
      github:
        secret: "secret101"
```

```
  - type: "Generic"
    generic:
      secret: "secret101"
  -
    type: "ImageChange"
source: 4
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
strategy: 5
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
output: 6
  to:
    kind: "ImageStreamTag"
    name: "origin-ruby-sample:latest"
postCommit: 7
    script: "bundle exec rake test"
```

1. 此规格会创建一个名为 **ruby-sample-build** 的新 **BuildConfig**。
2. **runPolicy** 字段控制从此构建配置创建的构建能否同时运行。默认值为 **Serial**，即新构建将按顺序运行，而不是同时运行。
3. 您可以指定导致创建新构建的触发器的列表。
4. **source** 部分定义构建的来源。源类型决定主要的输入源，可以是 **Git**（指向代码库存储位置）、**Dockerfile**（从内联 Dockerfile 构建）或 **Binary**（接受二进制有效负载）。可以同时拥有多个源。有关每种源类型的更多信息，请参阅"创建构建输入"。
5. **strategy** 部分描述用于执行构建的构建策略。您可以在此处指定 **Source**、**Docker** 或 **Custom** 策略。本例使用 **ruby-20-centos7** 容器镜像， Source-to-image（S2I）用于应用程序构建。
6. 成功构建容器镜像后，它将被推送到 **output** 部分中描述的存储库。
7. **postCommit** 部分定义一个可选构建 hook。

2.3. 创建构建输入

通过以下小节查看构建输入的概述，并了解如何使用输入提供构建操作的源内容，以及如何使用构建环境和创建 secret。

2.3.1. 构建输入

构建输入提供构建操作的源内容。您可以使用以下构建输入在 OpenShift Container Platform 中提供源，它们按优先顺序列出：

- 内联 Dockerfile 定义
- 从现有镜像中提取内容
- Git 存储库

- 二进制（本地）输入
- 输入 secret
- 外部工件 (artifact)

您可以在单个构建中组合多个输入。但是，由于内联 Dockerfile 具有优先权，它可能会覆盖任何由其他输入提供的名为 Dockerfile 的文件。二进制（本地）和 Git 存储库是互斥的输入。

如果不希望在构建生成的最终应用程序镜像中提供构建期间使用的某些资源或凭证，或者想要消耗在 secret 资源中定义的值，您可以使用输入 secret。外部工件可用于拉取不以其他任一构建输入类型提供的额外文件。

在运行构建时：

1. 构造工作目录，并将所有输入内容放进工作目录中。例如，把输入 Git 存储库克隆到工作目录中，并且把由输入镜像指定的文件通过目标目录复制到工作目录中。
2. 构建过程将目录更改到 **contextDir**（若已指定）。
3. 内联 Dockerfile（若有）写入当前目录中。
4. 当前目录中的内容提供给构建过程，供 Dockerfile、自定义构建器逻辑或 **assemble** 脚本引用。这意味着，构建会忽略所有驻留在 **contextDir** 之外的输入内容。

以下源定义示例包括多种输入类型，以及它们如何组合的说明。如需有关如何定义各种输入类型的更多详细信息，请参阅每种输入类型的具体小节。

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git (1)
    ref: "master"
  images:
  - from:
      kind: ImageStreamTag
      name: myinputimage:latest
      namespace: mynamespace
    paths:
    - destinationDir: app/dir/injected/dir (2)
      sourcePath: /usr/lib/somefile.jar
  contextDir: "app/dir" (3)
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" (4)
```

(1) 要克隆到构建的工作目录中的存储库。

(2) 来自 **myinputimage** 的 **/usr/lib/somefile.jar** 存储在 **<workingdir> /app/dir/injected/dir** 中。

(3) 构建的工作目录将变为 **<original_workingdir>/app/dir**。

(4) **<original_workingdir>/app/dir** 中创建了含有此内容的 Dockerfile，并覆盖具有该名称的任何现有文件。

2.3.2. Dockerfile 源

提供 **dockerfile** 值时，此字段的内容将写到磁盘上，存为名为 **Dockerfile** 的文件。这是处理完其他输入源之后完成的；因此，如果输入源存储库的根目录中包含 Dockerfile，它会被此内容覆盖。

源定义是 **BuildConfig** 的 **spec** 部分的一部分：

```
source:
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" (1)
```

(1) **dockerfile** 字段包含要构建的内联 Dockerfile。

其他资源

- 此字段的典型用途是为 Docker 策略构建提供 Dockerfile。

2.3.3. 镜像源

您可以使用镜像向构建过程添加其他文件。输入镜像的引用方式与定义 **From** 和 **To** 镜像目标的方式相同。这意味着可以引用容器镜像和镜像流标签。在使用镜像时，必须提供一个或多个路径对，以指示要复制镜像的文件或目录的路径以及构建上下文中要放置它们的目的地。

源路径可以是指定镜像内的任何绝对路径。目的地必须是相对目录路径。构建时会加载镜像，并将指定的文件和目录复制到构建过程上下文目录中。这与源存储库内容要克隆到的目录相同。如果源路径以 **/.** 结尾，则复制目录的内容，但不在目的地上创建该目录本身。

镜像输入在 **BuildConfig** 的 **source** 定义中指定：

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git
    ref: "master"
  images: (1)
  - from: (2)
      kind: ImageStreamTag
      name: myinputimage:latest
      namespace: mynamespace
    paths: (3)
    - destinationDir: injected/dir (4)
      sourcePath: /usr/lib/somefile.jar (5)
  - from:
      kind: ImageStreamTag
      name: myotherinputimage:latest
      namespace: myothernamespace
    pullSecret: mysecret (6)
    paths:
    - destinationDir: injected/dir
      sourcePath: /usr/lib/somefile.jar
```

(1) 由一个或多个输入镜像和文件组成的数组。

(2) 对包含要复制的文件的镜像的引用。

(3) 源/目标路径的数组。

4 相对于构建过程能够处理文件的构建根目录的目录。

5 要从所引用镜像中复制文件的位置。

6 提供的可选 secret，如需要凭证才能访问输入镜像。

注意

如果您的集群使用 **ImageContentSourcePolicy** 对象来配置存储库镜像，则只能将全局 pull secret 用于镜像 registry。您不能在项目中添加 pull secret。

另外，如果输入镜像需要 pull secret，您可以将 pull secret 链接到构建所使用的服务帐户。默认情况下，构建使用 **builder** 服务帐户。如果 secret 包含与托管输入镜像的存储库匹配的凭证，pull secret 会自动添加到构建中。要将 pull secret 连接到构建使用的服务帐户，请运行：

```
$ oc secrets link builder dockerhub
```

注意

使用自定义策略的构建不支持此功能。

2.3.4. Git 源

指定之后，从提供的位置获取源代码。

如果您提供内联 Dockerfile，它将覆盖 Git 存储库的 **contextDir** 中的 Dockerfile。

源定义是 **BuildConfig** 的 **spec** 部分的一部分：

```
source:
  git: 1
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
  contextDir: "app/dir" 2
  dockerfile: "FROM openshift/ruby-22-centos7\nUSER example" 3
```

1 **git** 字段包含源代码的远程 Git 存储库的 URI (Uniform Resource Identifier)。您必须指定 **ref** 字段的值来签出特定的 Git 引用。有效的 **ref** 可以是 SHA1 标签或分支名称。**ref** 字段的默认值为 **master**。

2 **contextDir** 字段允许您覆盖源代码存储库中构建查找应用程序源代码的默认位置。如果应用程序位于子目录中，您可以使用此字段覆盖默认位置（根文件夹）。

3 如果提供可选的 **dockerfile** 字段，它应该是包含 Dockerfile 的字符串，此文件将覆盖源存储库中可能存在的任何 Dockerfile。

如果 **ref** 字段注明拉取请求，则系统将使用 **git fetch** 操作，然后 checkout **FETCH_HEAD**。

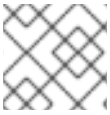
如果未提供 **ref** 值，OpenShift Container Platform 将执行浅克隆 (**--depth=1**)。这时，仅下载与默认分支（通常为 **master**）上最近提交相关联的文件。这将使存储库下载速度加快，但不会有完整的提交历史记录。要执行指定存储库的默认分支的完整 **git clone**，请将 **ref** 设置为默认分支的名称（如 **main**）。

警告

如果 Git 克隆操作要经过执行中间人 (MITM) TLS 劫持或重新加密被代理连接的代理，该操作不起作用。

2.3.4.1. 使用代理

如果 Git 存储库只能使用代理访问，您可以在构建配置的 **source** 部分中定义要使用的代理。您可以同时配置要使用的 HTTP 和 HTTPS 代理。两个字段都是可选的。也可以在 **NoProxy** 字段中指定不应执行代理的域。



注意

源 URI 必须使用 HTTP 或 HTTPS 协议才可以正常工作。

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```

注意

对于 Pipeline 策略构建，因为 Jenkins Git 插件当前限制的缘故，通过 Git 插件执行的任何 Git 操作都不会利用 **BuildConfig** 中定义的 HTTP 或 HTTPS 代理。Git 插件将仅使用 Plugin Manager 面板上 Jenkins UI 中配置的代理。然后，在所有任务中，此代理都会被用于 Jenkins 内部与 git 的所有交互。

其他资源

- 您可以在 JenkinsBehindProxy 上找到有关如何通过 Jenkins UI 配置代理的说明。

2.3.4.2. 源克隆 secret

构建器 pod 需要访问定义为构建源的任何 Git 存储库。源克隆 secret 为构建器 pod 提供了通常无权访问的资源的访问权限，例如私有存储库或具有自签名或不可信 SSL 证书的存储库。

支持以下源克隆 secret 配置：

- .gitconfig 文件
- 基本身份验证
- SSH 密钥身份验证
- 可信证书颁发机构

注意

您还可以组合使用这些配置来满足特定的需求。

2.3.4.2.1. 自动把源克隆 secret 添加到构建配置

创建 **BuildConfig**，OpenShift Container Platform 可以自动填充其源克隆 secret 引用。这会使生成的构建自动使用存储在引用的 secret 中的凭证与远程 Git 存储库进行身份验证，而无需进一步配置。

若要使用此功能，包含 Git 存储库凭证的一个 secret 必须存在于稍后创建 **BuildConfig** 的命名空间中。此 secret 必须包含前缀为 **build.openshift.io/source-secret-match-uri-** 的一个或多个注解。这些注解中的每一个值都是统一资源标识符（URI）模式，其定义如下。如果 **BuildConfig** 是在没有源克隆 secret 引用的前提下创建的，并且其 Git 源 URI 与 secret 注解中的 URI 模式匹配，OpenShift Container Platform 将自动在 **BuildConfig** 插入对该 secret 的引用。

先决条件

URI 模式必须包含：

- 有效的方案包括： ***://**、**git://**、**http://**、**https://** 或 **ssh://**
- 一个主机：*,或一个有效的主机名或 IP 地址（可选）在之前使用 ***。
- 一个路径： /*，或 /（后面包括任意字符并可以包括 * 字符）

在上述所有内容中，* 字符被认为是通配符。

重要

URI 模式必须与符合 RFC3986 的 Git 源 URI 匹配。不要在 URI 模式中包含用户名（或密码）组件。

例如，如果使用 **ssh://git@bitbucket.atlassian.com:7999/ATLASSIAN jira.git** 作为 git 存储库 URL，则源 secret 必须指定为 **ssh://bitbucket.atlassian.com:7999/***（而非 **ssh://git@bitbucket.atlassian.com:7999/***）。

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=ssh://bitbucket.atlassian.com:7999/*'
```

流程

如果多个 secret 与特定 **BuildConfig** 的 Git URI 匹配，OpenShift Container Platform 会选择匹配最多的 secret。这可以实现下例中所示的基本覆盖。

以下片段显示了两个部分源克隆 secret，第一个匹配通过 HTTPS 访问的 **mycorp.com** 域中的任意服务器，第二个则覆盖对服务器 **mydev1.mycorp.com** 和 **mydev2.mycorp.com** 的访问：

```
kind: Secret
apiVersion: v1
metadata:
  name: matches-all-corporate-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://*.mycorp.com/*
data:
  ...
---
```

```
kind: Secret
apiVersion: v1
metadata:
  name: override-for-my-dev-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://mydev1.mycorp.com/*
    build.openshift.io/source-secret-match-uri-2: https://mydev2.mycorp.com/*
data:
  ...
```

- 使用以下命令将 **build.openshift.io/source-secret-match-uri-** 注解添加到预先存在的 secret：

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=https://*.mycorp.com/*'
```

2.3.4.2.2. 手动添加源克隆 secret

通过将 **sourceSecret** 字段添加到 **BuildConfig** 中的 **source** 部分，并将它设置为您创建的 secret 的名称，可以手动将源克隆 secret 添加到构建配置中。在本例中，是 **basicsecret**。

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
  source:
    git:
      uri: "https://github.com/user/app.git"
    sourceSecret:
      name: "basicsecret"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "python-33-centos7:latest"
```

流程

您还可以使用 **oc set build-secret** 命令在现有构建配置中设置源克隆 secret。

- 要在现有构建配置上设置源克隆 secret，请输入以下命令：

```
$ oc set build-secret --source bc/sample-build basicsecret
```

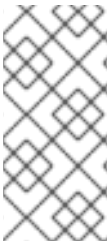
2.3.4.2.3. 从 .gitconfig 文件创建 secret

如果克隆应用程序要依赖于 **.gitconfig** 文件，您可以创建包含它的 secret。将它添加到 builder 服务帐户中，再添加到您的 **BuildConfig** 。

流程

- 从 **.gitconfig** 文件创建 secret：

```
$ oc create secret generic <secret_name> --from-file=<path/to/.gitconfig>
```



注意

如果 **.gitconfig** 文件的 **http** 部分设置了 **sslVerify=false**，则可以关闭 iVSSL 验证：

```
[http]
        sslVerify=false
```

2.3.4.2.4. 从 .gitconfig 文件为安全 Git 创建 secret

如果 Git 服务器使用双向 SSL 和用户名加密码进行保护，您必须将证书文件添加到源构建中，并在 **.gitconfig** 文件中添加对证书文件的引用。

先决条件

- 您必须具有 Git 凭证。

流程

将证书文件添加到源构建中，并在 **gitconfig** 文件中添加对证书文件的引用。

1. 将 **client.crt**、**cacert.crt** 和 **client.key** 文件添加到应用程序源代码的 **/var/run/secrets/openshift.io/source/** 目录中。
2. 在服务器的 **.gitconfig** 文件中，添加下例中所示的 **[http]** 部分：

```
# cat .gitconfig
```

输出示例

```
[user]
        name = <name>
        email = <email>
[http]
        sslVerify = false
        sslCert = /var/run/secrets/openshift.io/source/client.crt
        sslKey = /var/run/secrets/openshift.io/source/client.key
        sslCaInfo = /var/run/secrets/openshift.io/source/cacert.crt
```

3. 创建 secret：

```
$ oc create secret generic <secret_name> \
--from-literal=username=<user_name> \ 1
--from-literal=password=<password> \ 2
--from-file=.gitconfig=.gitconfig \
--from-file=client.crt=/var/run/secrets/openshift.io/source/client.crt \
--from-file=cacert.crt=/var/run/secrets/openshift.io/source/cacert.crt \
--from-file=client.key=/var/run/secrets/openshift.io/source/client.key
```

1 用户的 Git 用户名。

2 此用户的密码。

重要

为了避免必须再次输入密码，需要在构建中指定 source-to-image（S2I）镜像。但是，如果无法克隆存储库，您仍然必须指定用户名和密码才能推进构建。

其他资源

- 应用程序源代码中的 **/var/run/secrets/openshift.io/source/** 文件夹。

2.3.4.2.5. 从源代码基本身份验证创建 secret

基本身份验证需要 **--username** 和 **--password** 的组合，或者令牌方可与软件配置管理（SCM）服务器进行身份验证。

先决条件

- 用于访问私有存储库的用户名和密码。

流程

1. 在使用 **--username** 和 **--password** 访问私有存储库前首先创建 secret:

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --type=kubernetes.io/basic-auth
```

2. 使用令牌创建基本身份验证 secret：

```
$ oc create secret generic <secret_name> \
    --from-literal=password=<token> \
    --type=kubernetes.io/basic-auth
```

2.3.4.2.6. 从源代码 SSH 密钥身份验证创建 secret

基于 SSH 密钥的身份验证需要 SSH 私钥。

存储库密钥通常位于 **$HOME/.ssh/** 目录中，但默认名称为 **id_dsa.pub**、**id_ecdsa.pub**、**id_ed25519.pub** 或 **id_rsa.pub**。

流程

1. 生成 SSH 密钥凭证：

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

注意

使用带有密语保护的 SSH 密钥会导致 OpenShift Container Platform 无法进行构建。提示输入密语（passphrase）时，请将其留空。

创建两个文件：公钥和对应的私钥（**id_dsa**、**id_ecdsa**、**id_ed25519** 或 **id_rsa** 之一）。这两项就位后，请查阅源代码控制管理 (SCM) 系统的手册来了解如何上传公钥。私钥用于访问您的私有存储库。

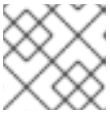
2. 在使用 SSH 密钥访问私有存储库之前，先创建 secret：

```
$ oc create secret generic <secret_name> \
    --from-file=ssh-privatekey=<path/to/ssh/private/key> \
    --from-file=<path/to/known_hosts> \ 1
    --type=kubernetes.io/ssh-auth
```

1 可选：添加此字段可启用严格的服务器主机密钥检查。

警告

在创建 secret 时跳过 **known_hosts** 文件会使构建容易受到中间人 (MITM) 攻击的影响。



注意

确保 **known_hosts** 文件中包含源代码主机条目。

2.3.4.2.7. 从源代码可信证书颁发机构创建 secret

在 Git 克隆操作期间受信任的 TLS 证书颁发机构（CA）集合内置于 OpenShift Container Platform 基础结构镜像中。如果 Git 服务器使用自签名证书或由镜像不信任的颁发机构签名的证书，您可以创建包含证书的 secret 或者禁用 TLS 验证。

如果您为 CA 证书创建 secret，OpenShift Container Platform 会在 Git 克隆操作过程中使用它来访问您的 Git 服务器。使用此方法比禁用 Git 的 SSL 验证要安全得多，后者接受所出示的任何 TLS 证书。

流程

使用 CA 证书文件创建 secret。

1. 如果您的 CA 使用中间证书颁发机构，请合并 **ca.crt** 文件中所有 CA 的证书。使用以下命令：

```
$ cat intermediateCA.crt intermediateCA.crt rootCA.crt > ca.crt
```

a. 创建 secret：

```
$ oc create secret generic mycert --from-file=ca.crt=</path/to/file> 1
```

1 您必须使用密钥名称 **ca.crt**。

2.3.4.2.8. 源 secret 组合

您可以组合使用不同的源克隆 secret 创建方法来满足特定的需求。

2.3.4.2.8.1. 使用 .gitconfig 文件创建基于 SSH 的身份验证 secret

您可以组合不同的方法开创建源克隆 secret 以满足特定的需求，例如使用 **.gitconfig** 文件的基于 SSH 的身份验证 secret。

先决条件

- SSH 身份验证
- .gitconfig 文件

流程

- 使用 **.gitconfig** 文件创建基于 SSH 的身份验证 secret：

```
$ oc create secret generic <secret_name> \
    --from-file=ssh-privatekey=<path/to/ssh/private/key> \
    --from-file=<path/to/.gitconfig> \
    --type=kubernetes.io/ssh-auth
```

2.3.4.2.8.2. 创建组合了 .gitconfig 文件和 CA 证书的 secret

您可以组合使用不同的源克隆 secret 创建方法来满足特定的需求，例如组合了 **.gitconfig** 文件和 CA 证书的 Secret。

先决条件

- .gitconfig 文件
- CA 证书

流程

- 创建组合了 **.gitconfig** 文件和 CA 证书的 secret：

```
$ oc create secret generic <secret_name> \
    --from-file=ca.crt=<path/to/certificate> \
    --from-file=<path/to/.gitconfig>
```

2.3.4.2.8.3. 使用 CA 证书创建基本身份验证 secret

您可以组合使用不同的源克隆 secret 创建方法来满足特定的需求，例如组合了基本身份验证和 CA 证书的 secret。

先决条件

- 基本身份验证凭证
- CA 证书

流程

- 使用 CA 证书创建基本身份验证 secret：

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=ca-cert=</path/to/file> \
    --type=kubernetes.io/basic-auth
```

2.3.4.2.8.4. 使用 .gitconfig 文件创建基本身份验证 secret

您可以组合使用不同的源克隆 secret 创建方法来满足特定的需求，例如组合了基本身份验证和 **.gitconfig** 文件的 secret。

先决条件

- 基本身份验证凭证
- **.gitconfig** 文件

流程

- 使用 **.gitconfig** 文件创建基本身份验证 secret：

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=</path/to/.gitconfig> \
    --type=kubernetes.io/basic-auth
```

2.3.4.2.8.5. 使用 .gitconfig 文件和 CA 证书创建基本身份验证 secret

您可以组合使用不同的源克隆 secret 创建方法来满足特定的需求，例如组合了基本身份验证、**.gitconfig** 文件和证书颁发机构（CA）证书的 Secret。

先决条件

- 基本身份验证凭证
- **.gitconfig** 文件
- CA 证书

流程

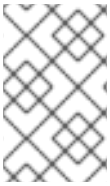
- 使用 **.gitconfig** 文件和 CA 证书创建基本身份验证 secret：

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=</path/to/.gitconfig> \
    --from-file=ca-cert=</path/to/file> \
    --type=kubernetes.io/basic-auth
```

2.3.5. 二进制（本地）来源

从本地文件系统流传输内容到构建器称为 **Binary** 类型构建。对于此类构建，**BuildConfig.spec.source.type** 的对应值为 **Binary**。

这种源类型的独特之处在于，它仅基于您对 **oc start-build** 的使用而加以利用。



注意

二进制类型构建需要从本地文件系统流传输内容，因此无法自动触发二进制类型构建，如镜像更改触发器。这是因为无法提供二进制文件。同样，您无法从 web 控制台启动二进制类型构建。

要使用二进制构建，请使用以下选项之一调用 **oc start-build**：

- **--from-file**：指定的文件内容作为二进制流发送到构建器。您还可以指定文件的 URL。然后，构建器将数据存储在构建上下文顶端的同名文件中。
- **--from-dir** 和 **--from-repo**：内容存档下来，并作为二进制流发送给构建器。然后，构建器在构建上下文目录中提取存档的内容。使用 **--from-dir** 时，您还可以指定提取的存档的 URL。
- **--from-archive**：指定的存档发送到构建器，并在构建器上下文目录中提取。此选项与 **--from-dir** 的行为相同；只要这些选项的参数是目录，就会首先在主机上创建存档。

在上方列出的每种情形中：

- 如果 **BuildConfig** 已经定义了 **Binary** 源类型，它会有效地被忽略并且替换成客户端发送的内容。
- 如果 **BuildConfig** 定义了 **Git** 源类型，则会动态禁用它，因为 **Binary** 和 **Git** 是互斥的，并且二进制流中提供给构建器的数据将具有优先权。

您可以将 HTTP 或 HTTPS 方案的 URL 传递给 **--from-file** 和 **--from-archive**，而不传递文件名。将 **--from-file** 与 URL 结合使用时，构建器镜像中文件的名称由 web 服务器发送的 **Content-Disposition** 标头决定，如果该标头不存在，则由 URL 路径的最后一个组件决定。不支持任何形式的身份验证，也无法使用自定义 TLS 证书或禁用证书验证。

使用 **oc new-build --binary =true** 时，该命令可确保强制执行与二进制构建关联的限制。生成的 **BuildConfig** 具有 **Binary** 源类型，这意味着为此 **BuildConfig** 运行构建的唯一有效方法是使用 **oc start-build** 和其中一个 **--from** 选项来提供必需的二进制数据。

Dockerfile 和 **contextDir** 源选项对二进制构建具有特殊含义。

Dockerfile 可以与任何二进制构建源一起使用。如果使用 Dockerfile 且二进制流是存档，则其内容将充当存档中任何 Dockerfile 的替代 Dockerfile。如果将 Dockerfile 与 **--from-file** 参数搭配 使用，且文件参数命名为 Dockerfile，则 Dockerfile 的值将替换二进制流中的值。

如果是二进制流封装提取的存档内容，**contextDir** 字段的值将解释为存档中的子目录，并且在有效时，构建器将在执行构建之前更改到该子目录。

2.3.6. 输入 secret 和配置映射

重要

要防止输入 secret 和配置映射的内容出现在构建输出容器镜像中，请使用 Docker 构建和源至镜像构建策略中的构建 卷。

有时候，构建操作需要凭证或其他配置数据才能访问依赖的资源，但又不希望将这些信息放在源代码控制中。您可以定义输入 secret 和输入配置映射。

例如，在使用 Maven 构建 Java 应用程序时，可以设置通过私钥访问的 Maven Central 或 JCenter 的私有镜像。要从该私有镜像下载库，您必须提供以下内容：

1. 配置了镜像的 URL 和连接设置的 **settings.xml** 文件。
2. 设置文件中引用的私钥，例如 ~/**.ssh/id_rsa**。

为安全起见，不应在应用程序镜像中公开您的凭证。

示例中描述的是 Java 应用程序，但您可以使用相同的方法将 SSL 证书添加到 **/etc/ssl/certs** 目录，以及添加 API 密钥或令牌、许可证文件等。

2.3.6.1. 什么是 secret？

Secret 对象类型提供了一种机制来保存敏感信息，如密码、OpenShift Container Platform 客户端配置文件、**dockercfg** 文件和私有源存储库凭证等。secret 将敏感内容与 Pod 分离。您可以使用卷插件将 secret 信息挂载到容器中，系统也可以使用 secret 代表 Pod 执行操作。

YAML Secret 对象定义

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque (1)
data: (2)
  username: <username> (3)
  password: <password>
stringData: (4)
  hostname: myapp.mydomain.com (5)
```

(1) 指示 secret 的键和值的结构。

(2) **data** 字段中允许的键格式必须符合 Kubernetes 标识符术语表中 **DNS_SUBDOMAIN** 值的规范。

(3) 与 **data** 映射中键关联的值必须采用 base64 编码。

(4) **stringData** 映射中的条目将转换为 base64，然后该条目将自动移动到 **data** 映射中。此字段是只读的。这个值只能由 **data** 字段返回。

(5) 与 **stringData** 映射中键关联的值由纯文本字符串组成。

2.3.6.1.1. secret 的属性

主要属性包括：

- Secret 数据可以独立于其定义来引用。
- Secret 数据卷由临时文件工具 (tmpfs) 支持，永远不会停留在节点上。

- secret 数据可以在命名空间内共享。

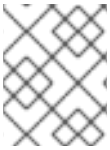
2.3.6.1.2. secret 的类型

type 字段中的值指明 secret 的键名称和值的结构。此类型可用于强制使 secret 对象中存在用户名和密钥。如果您不想进行验证，请使用 **opaque** 类型，这也是默认类型。

指定以下一种类型来触发最小服务器端验证，确保 secret 数据中存在特定的键名称：

- **kubernetes.io/service-account-token**。使用服务帐户令牌。
- **kubernetes.io/dockercfg**。将 **.dockercfg** 文件用于所需的 Docker 凭证。
- **kubernetes.io/dockerconfigjson**。将 **.docker/config.json** 文件用于所需的 Docker 凭证。
- **kubernetes.io/basic-auth**。与基本身份验证搭配使用。
- **kubernetes.io/ssh-auth**。搭配 SSH 密钥身份验证使用。
- **kubernetes.io/tls**。搭配 TLS 证书颁发机构使用。

如果不想进行验证，设置 **type= Opaque**。这意味着，secret 不声明符合键名称或值的任何约定。**opaque** secret 允许使用无结构 **key:value** 对，可以包含任意值。



注意

您可以指定其他任意类型，如 **example.com/my-secret-type**。这些类型不在服务器端强制执行，但代表 secret 的创建者意在符合该类型的键/值要求。

2.3.6.1.3. 更新 secret

当修改 secret 的值时，已在被运行的 Pod 使用的 secret 值不会被动态更新。要更改 secret，必须删除原始 pod 并创建一个新 pod，在某些情况下，具有相同的 **PodSpec**。

更新 secret 遵循与部署新容器镜像相同的工作流。您可以使用 **kubectl rolling-update** 命令。

secret 中的 **resourceVersion** 值不在引用时指定。因此，如果在 pod 启动的同时更新 secret，则将不能定义用于 pod 的 secret 版本。

注意

目前，无法检查 Pod 创建时使用的 secret 对象的资源版本。按照计划 Pod 将报告此信息，以便控制器可以重启使用旧 **resourceVersion** 的 Pod。在此期间，请勿更新现有 secret 的数据，而应创建具有不同名称的新数据。

2.3.6.2. 创建 secret

您必须先创建 secret，然后创建依赖于此 secret 的 Pod。

在创建 secret 时：

- 使用 secret 数据创建 secret 对象。
- 更新 pod 的服务帐户以允许引用该 secret。
- 创建以环境变量或文件（使用 **secret** 卷）形式消耗 secret 的 pod。

流程

- 使用创建命令从 JSON 或 YAML 文件创建 secret 对象：

```
$ oc create -f <filename>
```

例如，您可以从本地的 **.docker/config.json** 文件创建一个 secret：

```
$ oc create secret generic dockerhub \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

此命令将生成名为 **dockerhub** 的 secret JSON 规格并创建该对象。

YAML Opaque Secret 对象定义

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque 1
data:
  username: <username>
  password: <password>
```

1 指定一个 *opaque* secret。

Docker 配置 JSON 文件对象定义

```
apiVersion: v1
kind: Secret
metadata:
  name: aregistrykey
  namespace: myapps
type: kubernetes.io/dockerconfigjson 1
data:

.dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXIzCg== 2
```

1 指定该 secret 使用 docker 配置 JSON 文件。

2 base64 编码的 docker 配置 JSON 文件

2.3.6.3. 使用 secret

创建 secret 后，可以创建一个 Pod 来引用您的 secret，再获取日志，然后删除 Pod。

流程

1. 创建要引用您的 secret 的 Pod：

```
$ oc create -f <your_yaml_file>.yaml
```

2. 获取日志：

```
$ oc logs secret-example-pod
```

3. 删除 Pod：

```
$ oc delete pod secret-example-pod
```

其他资源

- 带有 secret 数据的 YAML 文件示例：

  将创建四个文件的 secret 的 YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: <username> 1
  password: <password> 2
stringData:
  hostname: myapp.mydomain.com 3
  secret.properties: |- 4
    property1=valueA
    property2=valueB
```

1 文件包含已解码的值。

2 文件包含已解码的值。

3 文件包含提供的字符串。

4 文件包含提供的数据。

  pod 的 YAML 使用 secret 数据填充卷中的文件

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
          # name must match the volume name below
          - name: secret-volume
            mountPath: /etc/secret-volume
```

```
        readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never
```

pod 的 YAML 使用 secret 数据填充环境变量

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username
  restartPolicy: Never
```

一个 Build Config 的 YAML 定义，在环境变量中使用 secret 数据

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username
```

2.3.6.4. 添加输入 secret 和配置映射

要向构建提供凭证和其他配置数据，而不将其放在源控制中，您可以定义输入 secret 和输入配置映射。

在某些情况下，构建操作需要凭证或其他配置数据才能访问依赖的资源。要使该信息在不置于源控制中的情况下可用，您可以定义输入 secret 和输入配置映射。

流程

将输入 secret 和配置映射添加到现有的 **BuildConfig** 对象中：

1. 如果 **ConfigMap** 对象不存在，则进行创建：

```
$ oc create configmap settings-mvn \
    --from-file=settings.xml=<path/to/settings.xml>
```

这会创建一个名为 **settings-mvn** 的新配置映射，其中包含 **settings.xml** 文件的纯文本内容。

提示

您还可以应用以下 YAML 来创建配置映射：

```
apiVersion: core/v1
kind: ConfigMap
metadata:
  name: settings-mvn
data:
  settings.xml: |
    <settings>
    … # Insert maven settings here
    </settings>
```

2. 如果 **Secret** 对象不存在，则进行创建：

```
$ oc create secret generic secret-mvn \
    --from-file=ssh-privatekey=<path/to/.ssh/id_rsa>
    --type=kubernetes.io/ssh-auth
```

这会创建一个名为 **secret-mvn** 的新 secret，其包含 **id_rsa** 私钥的 base64 编码内容。

提示

您还可以应用以下 YAML 来创建输入 secret：

```
apiVersion: core/v1
kind: Secret
metadata:
  name: secret-mvn
type: kubernetes.io/ssh-auth
data:
  ssh-privatekey: |
    # Insert ssh private key, base64 encoded
```

3. 将配置映射和 secret 添加到现有 **BuildConfig** 对象的 **source** 部分中：

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
  secrets:
    - secret:
        name: secret-mvn
```

要在新 **BuildConfig** 对象中包含 secret 和配置映射，请运行以下命令：

```
$ oc new-build \
    openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
    --context-dir helloworld --build-secret “secret-mvn” \
    --build-config-map "settings-mvn"
```

在构建期间，**settings.xml** 和 **id_rsa** 文件将复制到源代码所在的目录中。在 OpenShift Container Platform S2I 构建器镜像中，这是镜像的工作目录，使用 **Dockerfile** 中的 **WORKDIR** 指令设置。如果要指定其他目录，请在定义中添加 **destinationDir**：

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
      destinationDir: ".m2"
  secrets:
    - secret:
        name: secret-mvn
      destinationDir: ".ssh"
```

您还可以指定创建新 **BuildConfig** 对象时的目标目录：

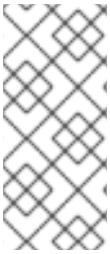
```
$ oc new-build \
    openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
    --context-dir helloworld --build-secret “secret-mvn:.ssh” \
    --build-config-map "settings-mvn:.m2"
```

在这两种情况下，**settings.xml** 文件都添加到构建环境的 **./.m2** 目录中，而 **id_rsa** 密钥则添加到 **./.ssh**目录中。

2.3.6.5. Source-to-Image 策略

采用 **Source** 策略时，所有定义的输入 secret 都复制到对应的 **destinationDir** 中。如果 **destinationDir** 留空，则 secret 会放置到构建器镜像的工作目录中。

当 **destinationDir** 是一个相对路径时，使用相同的规则。secret 放置在相对于镜像工作目录的路径中。如果构建器镜像中不存在 **destinationDir** 路径中的最终目录，则会创建该目录。**destinationDir** 中的所有上述目录都必须存在，否则会发生错误。



注意

输入 secret 将以全局可写（具有 **0666** 权限）形式添加，并且在执行 **assemble** 脚本后其大小会被截断为零。也就是说，生成的镜像中会包括这些 secret 文件，但出于安全原因，它们将为空。

assemble 脚本完成后不会截断输入配置映射。

2.3.6.6. Docker 策略

采用 docker 策略时，您可以使用 Dockerfile 中的 **ADD** 和 **COPY** 指令，将所有定义的输入 secret 添加到容器镜像中。

如果没有为 secret 指定 **destinationDir**，则文件将复制到 Dockerfile 所在的同一目录中。如果将一个相对路径指定为 **destinationDir**，则 secret 复制到相对于 Dockerfile 位置的该目录中。这样，secret 文件可供 Docker 构建操作使用，作为构建期间使用的上下文目录的一部分。

引用 secret 和配置映射数据的 Dockerfile 示例

```
FROM centos/ruby-22-centos7

USER root
COPY ./secret-dir /secrets
COPY ./config /

# Create a shell script that will output secrets and ConfigMaps when the image is run
RUN echo '#!/bin/sh' > /input_report.sh
RUN echo '(test -f /secrets/secret1 && echo -n "secret1=" && cat /secrets/secret1)' >>
/input_report.sh
RUN echo '(test -f /config && echo -n "relative-configMap=" && cat /config)' >> /input_report.sh
RUN chmod 755 /input_report.sh

CMD ["/bin/sh", "-c", "/input_report.sh"]
```

重要

用户应该从最终的应用程序镜像中移除输入 secret，以便从该镜像运行的容器中不会存在这些 secret。但是，secret 仍然存在于它们添加到的层中的镜像本身内。这一移除是 Dockerfile 本身的一部分。

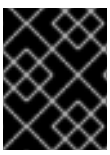
为防止输入 secret 和配置映射的内容出现在构建输出容器镜像中并完全避免此移除过程，请在 Docker 构建策略中使用构建卷。

2.3.6.7. Custom 策略

使用 Custom 策略时，所有定义的输入 secret 和配置映射都位于 **/var/run/secrets/openshift.io/build** 目录中的构建器容器中。自定义构建镜像必须正确使用这些 secret 和配置映射。使用 Custom 策略时，您可以按照 Custom 策略选项中所述定义 secret。

现有策略 secret 与输入 secret 之间没有技术差异。但是，构建器镜像可以区分它们并以不同的方式加以使用，具体取决于您的构建用例。

输入 secret 始终挂载到 **/var/run/secrets/openshift.io/build** 目录中，或您的构建器可以解析 **$BUILD** 环境变量（包含完整构建对象）。



重要

如果命名空间和节点上存在 registry 的 pull secret，构建会默认使用命名空间中的 pull secret。

2.3.7. 外部工件 (artifact)

建议不要将二进制文件存储在源存储库中。因此，您必须定义一个构建，在构建过程中拉取其他文件，如 Java **.jar** 依赖项。具体方法取决于使用的构建策略。

对于 Source 构建策略，必须在 **assemble** 脚本中放入适当的 shell 命令：

.s2i/bin/assemble 文件

```
#!/bin/sh
APP_VERSION=1.0
wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar
```

.s2i/bin/run 文件

```
#!/bin/sh
exec java -jar app.jar
```

对于 Docker 构建策略，您必须修改 Dockerfile 并通过 **RUN** 指令 调用 shell 命令：

Dockerfile 摘录

```
FROM jboss/base-jdk:8

ENV APP_VERSION 1.0
RUN wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar

EXPOSE 8080
CMD [ "java", "-jar", "app.jar" ]
```

在实践中，您可能希望将环境变量用于文件位置，以便要下载的具体文件能够使用 **BuildConfig** 中定义的环境变量来自定义，而不必更新 Dockerfile 或 **assemble** 脚本。

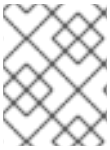
您可以选择不同方法来定义环境变量：

- 使用 **.s2i/environment** 文件（仅适用于 Source 构建策略）
- 在 **BuildConfig** 中设置
- 使用 **oc start-build --env** 明确提供（仅适用于手动触发的构建）

2.3.8. 将 docker 凭证用于私有容器镜像仓库

您可以为构建提供 **.docker/config.json** 文件，在文件中包含私有容器 registry 的有效凭证。这样，您可以将输出镜像推送到私有容器镜像 registry 中，或从需要身份验证的私有容器镜像 registry 中拉取构建器镜像。

您可以为同一 registry 中的多个存储库提供凭证，每个软件仓库都有特定于该 registry 路径的凭证。



注意

对于 OpenShift Container Platform 容器镜像 registry，这不是必需的，因为 OpenShift Container Platform 会自动为您生成 secret。

默认情况下，**.docker/config.json** 文件位于您的主目录中，并具有如下格式：

```
auths:
  index.docker.io/v1/: 1
```

```
    auth: "YWRfbGzhcGU6R2labnRib21ifTE=" 2
    email: "user@example.com" 3
  docker.io/my-namespace/my-user/my-image: 4
    auth: "GzhYWRGU6R2fbclabnRgbkSp=""
    email: "user@example.com"
  docker.io/my-namespace: 5
    auth: "GzhYWRGU6R2deesfrRgbkSp=""
    email: "user@example.com"
```

1. registry URL。
2. 加密的密码。
3. 用于登录的电子邮件地址。
4. 命名空间中的特定镜像的 URL 和凭证。
5. registry 命名空间的 URL 和凭证。

您可以定义多个容器镜像 registry，或在同一 registry 中定义多个存储库。或者，也可以通过运行 **docker login** 命令将身份验证条目添加到此文件中。如果文件不存在，则会创建此文件。

Kubernetes 提供**Secret** 对象，可用于存储配置和密码。

先决条件

- 您必须有一个 **.docker/config.json** 文件。

流程

1. 从本地 **.docker/config.json** 文件创建 secret：

```
$ oc create secret generic dockerhub \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

这将生成名为 **dockerhub** 的 secret 的 JSON 规格并创建该对象。

2. 将 **pushSecret** 字段添加到 **BuildConfig** 中的 **output** 部分，并将它设为您创建的 **secret** 的名称，上例中为 **dockerhub**：

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "private.registry.com/org/private-image:latest"
    pushSecret:
      name: "dockerhub"
```

您可以使用 **oc set build-secret** 命令在构建配置上设置推送 secret：

```
$ oc set build-secret --push bc/sample-build dockerhub
```

您还可以将 push secret 与构建使用的服务帐户链接，而不指定 **pushSecret** 字段。默认情况下，构建使用 **builder** 服务帐户。如果 secret 包含与托管构建输出镜像的存储库匹配的凭证，则 push secret 会自动添加到构建中。

```
$ oc secrets link builder dockerhub
```

3. 通过指定 **pullSecret** 字段（构建策略定义的一部分），从私有容器镜像 registry 拉取构建器容器镜像：

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "docker.io/user/private_repository"
    pullSecret:
      name: "dockerhub"
```

您可以使用 **oc set build-secret** 命令在构建配置上设置拉取 secret：

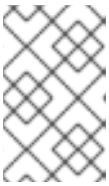
```
$ oc set build-secret --pull bc/sample-build dockerhub
```

注意

本例在 Source 构建中使用 **pullSecret**，但也适用于 Docker 构建和 Custom 构建。

您还可以将 pull secret 链接到构建使用的服务帐户，而不指定 **pullSecret** 字段。默认情况下，构建使用 **builder** 服务帐户。如果 secret 包含与托管构建的输入镜像的存储库匹配的凭证，pull secret 会自动添加到构建中。将 pull secret 链接到构建使用的服务帐户，而不指定 **pullSecret** 字段：

```
$ oc secrets link builder dockerhub
```



注意

您必须在 **BuildConfig** spec 中指定一个 **from** 镜像，才能利用此功能。由 **oc new-build** 或 **oc new-app** 生成的 Docker 策略构建在某些情况下可能无法进行这个操作。

2.3.9. 构建环境

与 Pod 环境变量一样，可以定义构建环境变量，在使用 Downward API 时引用其他源或变量。需要注意一些例外情况。

您也可以使用 **oc set env** 命令管理 **BuildConfig** 中定义的环境变量。

注意

不支持在构建环境变量中使用 **valueFrom** 引用容器资源，因为这种引用在创建容器之前解析。

2.3.9.1. 使用构建字段作为环境变量

您可以注入构建对象的信息，使用 **fieldPath** 环境变量源指定要获取值的字段的 **JsonPath**。

注意

Jenkins Pipeline 策略不支持将 **valueFrom** 语法用于环境变量。

流程

- 将 **fieldPath** 环境变量源设置为您有兴趣获取其值的字段的 **JsonPath**：

```
env:
  - name: FIELDREF_ENV
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
```

2.3.9.2. 使用 secret 作为环境变量

您可以使用 **valueFrom** 语法，将 secret 的键值作为环境变量提供。

重要

此方法在构建容器集控制台的输出中以纯文本形式显示机密。要避免这种情况，请使用输入 secret 和配置映射。

流程

- 要将 secret 用作环境变量，请设置 **valueFrom** 语法：

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: MYVAL
        valueFrom:
          secretKeyRef:
            key: myval
            name: mysecret
```

其他资源

- 输入 secret 和配置映射

2.3.10. 服务用（service serving）证书 secret

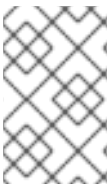
服务用证书 secret 旨在支持需要开箱即用证书的复杂中间件应用程序。它的设置与管理员工具为节点和 master 生成的服务器证书相同。

流程

要保护与您服务的通信，请让集群生成的签名的服务证书/密钥对保存在您的命令空间的 secret 中。

- 在服务上设置 **service.beta.openshift.io/serving-cert-secret-name** 注解，并将值设置为您要用于 secret 的名称。
  然后，您的 **PodSpec** 可以挂载该 secret。当它可用时，您的 Pod 会运行。该证书对内部服务 DNS 名称 **<service.name>.<service.namespace>.svc** 有效。

  证书和密钥采用 PEM 格式，分别存储在 **tls.crt** 和 **tls.key** 中。证书/密钥对在接近到期时自动替换。在 secret 的 **service.beta.openshift.io/expiry** 注解中查看过期日期，其格式为 RFC3339。



注意

在大多数情形中，服务 DNS 名称 **<service.name>.<service.namespace>.svc** 不可从外部路由。**<service.name>.<service.namespace>.svc** 的主要用途是集群内或服务内通信，也用于重新加密路由。

其他 pod 可以信任由集群创建的证书，这些证书只为内部 DNS 名称签名，方法是使用 pod 中自动挂载的 **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** 文件中的证书颁发机构（CA）捆绑包。

此功能的签名算法是 **x509.SHA256WithRSA**。要手动轮转，请删除生成的 secret。这会创建新的证书。

2.3.11. secret 限制

若要使用一个 secret，Pod 需要引用该 secret。可以通过三种方式将 secret 用于 Pod：

- 为容器产生环境变量。
- 作为挂载到一个或多个容器上的卷中的文件。
- 在拉取 Pod 的镜像时通过 kubelet 使用。

卷类型 secret 使用卷机制将数据作为文件写入到容器中。**imagePullSecrets** 使用服务帐户将 secret 自动注入到命名空间中的所有 Pod。

当模板包含 secret 定义时，模板使用提供的 secret 的唯一方法是确保验证 secret 卷源通过验证，并且指定的对象引用实际指向 **Secret** 类型的对象。因此，secret 需要在依赖它的任何 Pod 之前创建。确保这一点的最有效方法是通过使用服务帐户自动注入。

Secret API 对象驻留在命名空间中。它们只能由同一命名空间中的 pod 引用。

每个 secret 的大小限制为 1MB。这是为了防止创建可能会耗尽 apiserver 和 kubelet 内存的大型 secret。不过，创建大量较小的 secret 也可能耗尽内存。

2.4. 管理构建输出

以下小节提供了管理构建输出的概览和说明。

2.4.1. 构建输出

使用 docker 或 source-to-image（S2I）策略的构建会导致创建新的容器镜像。镜像而后被推送到由 **Build** 规格的 **output** 部分中指定的容器镜像 registry 中。

如果输出类型是 **ImageStreamTag**，则镜像将推送到集成的 OpenShift 镜像 registry 并在指定的镜像流中标记。如果输出类型为 **DockerImage**，则输出引用的名称将用作 docker push 规格。规格中可以包含 registry；如果没有指定 registry，则默认为 DockerHub。如果 Build 规格的 output 部分为空，则构建结

束时不推送镜像。

输出到 ImageStreamTag

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
```

输出到 docker Push 规格

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "my-registry.mycompany.com:5000/myimages/myimage:tag"
```

2.4.2. 输出镜像环境变量

Docker 和 source-to-image（S2I）策略构建设置输出镜像的以下环境变量：

变量	描述
OPENSHIFT_BUILD_NAME	构建的名称
OPENSHIFT_BUILD_NAMESPACE	构建的命名空间
OPENSHIFT_BUILD_SOURCE	构建的源 URL
OPENSHIFT_BUILD_REFERENCE	构建中使用的 Git 引用
OPENSHIFT_BUILD_COMMIT	构建中使用的源提交

此外，任何用户定义的环境变量（例如，使用 S2I 或 docker 策略选项配置的环境变量）也将是输出镜像环境变量列表的一部分。

2.4.3. 输出镜像标签

docker 和 Source-to-Image (S2I) 构建设置输出镜像的以下标签：

标签	描述
io.openshift.build.commit.author	构建中使用的源提交的作者
io.openshift.build.commit.date	构建中使用的源提交的日期
io.openshift.build.commit.id	构建中使用的源提交的哈希值

标签	描述
io.openshift.build.commit.message	构建中使用的源提交的消息
io.openshift.build.commit.ref	源中指定的分支或引用
io.openshift.build.source-location	构建的源 URL

您还可以使用 **BuildConfig.spec.output.imageLabels** 字段指定将应用到构建配置构建的每个镜像的自定义标签列表。

应用到所构建镜像的自定义标签

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "my-image:latest"
  imageLabels:
  - name: "vendor"
    value: "MyCompany"
  - name: "authoritative-source-url"
    value: "registry.mycompany.com"
```

2.5. 使用构建策略

以下小节定义了受支持的主要构建策略，以及它们的使用方法。

2.5.1. Docker 构建

OpenShift Container Platform 使用 Buildah 从 Dockerfile 构建容器镜像。有关使用 Dockerfile 构建容器镜像的更多信息，请参阅 Dockerfile 参考文档。

提示

如果使用 **buildArgs** 数组设置 Docker 构建参数，请参阅 Dockerfile 参考文档中 了解 ARG 和 FROM 如何交互。

2.5.1.1. 替换 Dockerfile FROM 镜像

您可以将 Dockerfile 中的 **FROM** 指令替换为 **BuildConfig** 对象的 **from**。如果 Dockerfile 使用多阶段构建，最后一个 **FROM** 指令中的镜像将被替换。

流程

将 Dockerfile 中的 **FROM** 指令替换为 **BuildConfig** 中的 **from**。

```
strategy:
  dockerStrategy:
    from:
```

```
      kind: "ImageStreamTag"
      name: "debian:latest"
```

2.5.1.2. 使用 Dockerfile 路径

默认情况下，docker 构建使用位于 **BuildConfig.spec.source.contextDir** 字段中指定的上下文的根目录下的 Dockerfile。

dockerfilePath 字段允许构建使用不同的路径来定位 Dockerfile，该路径相对于 **BuildConfig.spec.source.contextDir** 字段。它可以是不同于默认 Dockerfile 的其他文件名，如 **MyDockerfile**，也可以是子目录中 Dockerfile 的路径，如 **dockerfiles/app1/Dockerfile**。

流程

要通过构建的 **dockerfilePath** 字段使用不同的路径来定位 Dockerfile，请设置：

```
strategy:
  dockerStrategy:
    dockerfilePath: dockerfiles/app1/Dockerfile
```

2.5.1.3. 使用 Docker 环境变量

要将环境变量提供给 docker 构建过程和生成的镜像使用，您可以在构建配置的 **dockerStrategy** 定义中添加环境变量。

这里定义的环境变量作为单个 **ENV** Dockerfile 指令直接插入到 **FROM** 指令后，以便稍后可在 Dockerfile 内引用该变量。

流程

变量在构建期间定义并保留在输出镜像中，因此它们也会出现在运行该镜像的任何容器中。

例如，定义要在构建和运行时使用的自定义 HTTP 代理：

```
dockerStrategy:
...
  env:
    - name: "HTTP_PROXY"
      value: "http://myproxy.net:5187/"
```

您还可以使用 **oc set env** 命令管理构建配置中定义的环境变量。

2.5.1.4. 添加 Docker 构建参数

您可以使用 **buildArgs** 数组来设置 docker build 参数。构建参数将在构建启动时传递给 docker。

提示

请参阅 Dockerfile 参考文档中的 ARG 和 FROM 如何交互。

流程

要设置 docker 构建参数，请在 **buildArgs** 中添加条目，它位于 **BuildConfig** 对象的 **dockerStrategy** 定义中。例如：

```
dockerStrategy:
...
  buildArgs:
    - name: "foo"
      value: "bar"
```

注意

只支持 **name** 和 **value** 字段。**valueFrom** 字段上的任何设置都会被忽略。

2.5.1.5. 使用 docker 构建的 Squash 层

通常，Docker 构建会为 Dockerfile 中的每条指令都创建一个层。将 **imageOptimizationPolicy** 设置为 **SkipLayers**，可将所有指令合并到基础镜像顶部的单个层中。

流程

- 将 **imageOptimizationPolicy** 设置为 **SkipLayers**：

```
strategy:
  dockerStrategy:
    imageOptimizationPolicy: SkipLayers
```

2.5.1.6. 使用构建卷

您可以挂载构建卷，为运行的构建授予您不想在输出容器镜像中保留的信息的访问权限。

构建卷提供仅在构建时需要的敏感信息，如存储库凭据。构建卷与构建输入不同，后者的数据可以保留在输出容器镜像中。

构建卷的挂载点（运行中的构建从中读取数据）在功能上与 pod 卷挂载 类似。

先决条件

- 您已将输入 secret 和配置映射添加到 BuildConfig 对象中。

流程

- 在 **BuildConfig** 对象的 **dockerStrategy** 定义中，将任何构建卷添加到 **volumes** 数组中。例如：

```
spec:
  dockerStrategy:
    volumes:
      - name: secret-mvn (1)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh (2)
        source:
          type: Secret (3)
          secret:
            secretName: my-secret (4)
      - name: settings-mvn (5)
        mounts:
        - destinationPath: /opt/app-root/src/.m2 (6)
```

```
    source:
      type: ConfigMap (7)
      configMap:
        name: my-config (8)
    - name: my-csi-volume (9)
      mounts:
      - destinationPath: /opt/app-root/src/some_path (10)
      source:
        type: CSI (11)
        csi:
          driver: csi.sharedresource.openshift.io (12)
          readOnly: true (13)
          volumeAttributes: (14)
            attribute: value
```

(1) (5) (9) 必需。唯一的名称。

(2) (6) (10) 必需。挂载点的绝对路径。它不能包含 **..** 或 **:** 且不与构建器生成的目的地路径冲突。**/opt/app-root/src** 是许多支持 Red Hat S2I 的镜像的默认主目录。

(3) (7) (11) 必需。源类型，**ConfigMap**、**Secret** 或 **CSI**。

(4) (8) 必需。源的名称。

(12) 必需。提供临时 CSI 卷的驱动程序。

(13) 可选。如果为 true，这指示驱动程序提供只读卷。

(14) 可选。临时 CSI 卷的卷属性。如需支持的属性键和值，请参阅 CSI 驱动程序的文档。

注意

共享资源 CSI 驱动程序作为技术预览提供。

2.5.2. Source-to-image 构建

Source-to-Image (S2I) 是一种用于构建可重复生成的容器镜像的工具。它通过将应用程序源代码注入容器镜像并汇编新镜像来生成可随时运行的镜像。新镜像融合了基础镜像（构建器）和构建的源代码，并可搭配 **buildah run** 命令使用。S2I 支持递增构建，可重复利用以前下载的依赖项和过去构建的工件等。

2.5.2.1. 执行 source-to-image 增量构建

Source-to-image (S2I) 可以执行增量构建，也就是能够重复利用过去构建的镜像中的工件。

流程

- 要创建增量构建，请创建对策略定义进行以下修改：

```
strategy:
  sourceStrategy:
    from:
```

```
      kind: "ImageStreamTag"
      name: "incremental-image:latest" (1)
    incremental: true (2)
```

(1) 指定支持增量构建的镜像。请参考构建器镜像的文档，以确定它是否支持此行为。

(2) 此标志（flag）控制是否尝试增量构建。如果构建器镜像不支持增量构建，则构建仍将成功，但您会收到一条日志消息，指出增量构建因为缺少 **save-artifacts** 脚本而未能成功。

其他资源

- 如需有关如何创建支持增量构建的构建器镜像的信息，请参阅 S2I 要求。

2.5.2.2. 覆盖 source-to-image 构建器镜像脚本

您可以覆盖构建器镜像提供的 **assemble**、**run** 和 **save-artifacts** source-to-image(S2I)脚本。

流程

要覆盖构建器镜像提供的 **assemble**、**run** 和 **save-artifacts** S2I 脚本，请执行以下任一操作：

- 在应用程序源存储库的 **.s2i/bin** 目录中提供 **assemble**、**run** 或 **save-artifacts** 脚本。
- 提供包含脚本的目录的 URL，作为策略定义的一部分。例如：

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "builder-image:latest"
    scripts: "http://somehost.com/scripts_directory" (1)
```

(1) 此路径会将 **run**、**assemble** 和 **save-artifacts** 附加到其中。如果找到任何或所有脚本，将使用它们代替镜像中提供的同名脚本。

注意

位于 **scripts** URL 的文件优先于源存储库的 **.s2i/bin** 中的文件。

2.5.2.3. Source-to-image 环境变量

可以通过两种方式将环境变量提供给源构建过程和生成的镜像。环境文件和 BuildConfig 环境值。提供的变量将存在于构建过程和输出镜像中。

2.5.2.3.1. 使用 Source-to-image 环境文件

利用源代码构建，您可以在应用程序内设置环境值（每行一个），方法是在源存储库中的 **.s2i/environment** 文件中指定它们。此文件中指定的环境变量存在于构建过程和输出镜像。

如果您在源存储库中提供 **.s2i/environment** 文件，则 source-to-image(S2I)会在构建期间读取此文件。这允许自定义构建行为，因为 **assembe** 脚本可能会使用这些变量。

流程

例如，在构建期间禁用 Rails 应用程序的资产编译：

- 在 **.s2i/environment** 文件中添加 **DISABLE_ASSET_COMPILATION=true**。

除了构建之外，指定的环境变量也可以在运行的应用程序本身中使用。例如，使 Rails 应用程序在 **development** 模式而非 **production** 模式中启动：

- 在 **.s2i/environment** 文件中添加 **RAILS_ENV=development**。

使用镜像部分中提供了各个镜像支持的环境变量的完整列表。

2.5.2.3.2. 使用 Source-to-image 构建配置环境

您可以在构建配置的 **sourceStrategy** 定义中添加环境变量。这里定义的环境变量可在 **assemble** 脚本执行期间看到，也会在输出镜像中定义，使它们能够供 **run** 脚本和应用程序代码使用。

流程

- 例如，禁用 Rails 应用程序的资产编译：

```
sourceStrategy:
...
  env:
    - name: "DISABLE_ASSET_COMPILATION"
      value: "true"
```

其他资源

- “构建环境”部分提供了更多高级指导。
- 您还可以使用 **oc set env** 命令管理构建配置中定义的环境变量。

2.5.2.4. 忽略 source-to-image 源文件

Source-to-Image（S2I） 支持 **.s2iignore** 文件，该文件包含了需要被忽略的文件列表。构建工作目录中的文件（由各种输入源提供）若与 **.s2iignore** 文件中指定的文件匹配，将不会提供给 **assemble** 脚本使用。

2.5.2.5. 使用 Source-to-image 从源代码创建镜像

Source-to-Image (S2I) 是一种框架，它可以轻松地将应用程序源代码作为输入，生成可运行编译的应用程序的新镜像。

使用 S2I 构建可重复生成的容器镜像的主要优点是便于开发人员使用。作为构建器镜像作者，您必须理解两个基本概念，构建过程和 S2I 脚本，才能让您的镜像提供最佳的 S2I 性能。

2.5.2.5.1. 了解 source-to-image 构建过程

构建过程包含以下三个基本元素，这些元素组合成最终的容器镜像：

- 源
- Source-to-image(S2I)脚本
- 构建器镜像

S2I 生成带有构建器镜像的 Dockerfile 作为第一个 **FROM** 指令。然后，由 S2I 生成的 Dockerfile 会被传递给 Buildah。

2.5.2.5.2. 如何编写 Source-to-image 脚本

您可以使用任何编程语言编写 S2I 脚本，只要脚本可在构建器镜像中执行。S2I 支持多种提供 **assemble/run/save-artifacts** 脚本的选项。每次构建时按以下顺序检查所有这些位置：

1. 构建配置中指定的脚本。
2. 在应用程序源 **.s2i/bin** 目录中找到的脚本。
3. 在默认镜像 URL 中找到的带有 **io.openshift.s2i.scripts-url** 标签的脚本。

镜像中指定的 **io.openshift.s2i.scripts-url** 标签和构建配置中指定的脚本都可以采用以下形式之一：

- **image:///path_to_scripts_dir**：镜像中 S2I 脚本所处目录的绝对路径
- **file:///path_to_scripts_dir**：主机上 S2I 脚本所处目录的相对或绝对路径
- **http(s)://path_to_scripts_dir**：S2I 脚本所处目录的 URL

表 2.1. S2I 脚本

脚本	描述
assemble	**assemble** 用来从源代码构建应用程序工件，并将其放置在镜像内部的适当目录中的脚本。这个脚本是必需的。此脚本的工作流为： 1. 可选：恢复构建工件。如果要支持增量构建，确保同时定义了 **save-artifacts**。 2. 将应用程序源放在所需的位置。 3. 构建应用程序工件。 4. 将工件安装到适合它们运行的位置。
run	**run** 脚本将执行您的应用程序。这个脚本是必需的。
save-artifacts	**save-artifacts** 脚本将收集所有可加快后续构建过程的依赖项。这个脚本是可选的。例如： - 对于 Ruby，由 Bundler 安装的 **gem**。 - 对于 Java，**.m2** 内容。 这些依赖项会收集到一个 **tar** 文件中，并传输到标准输出。
usage	借助 **usage** 脚本，可以告知用户如何正确使用您的镜像。这个脚本是可选的。

脚本	描述
test/run	借助 **test/run** 脚本，可以创建一个进程来检查镜像是否正常工作。这个脚本是可选的。该流程的建议工作流是： 1. 构建镜像。 2. 运行镜像以验证 **usage** 脚本。 3. 运行 **s2i build** 以验证 **assemble** 脚本。 4. 可选：再次运行 **s2i build**，以验证 **save-artifacts** 和 **assemble** 脚本的保存和恢复工件功能。 5. 运行镜像，以验证测试应用程序是否正常工作。  **注意** 建议将 **test/run** 脚本构建的测试应用程序放置到镜像存储库中的 **test/test-app** 目录。

S2I 脚本示例

以下示例 S2I 脚本采用 Bash 编写。每个示例都假定其 **tar** 内容解包到 **/tmp/s2i** 目录中。

assemble 脚本：

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run 脚本：

```
#!/bin/bash

# run the application
/opt/application/run.sh
```

save-artifacts 脚本：

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage 脚本：

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

其他资源

- S2I 镜像创建教程

2.5.2.6. 使用构建卷

您可以挂载构建卷，为运行的构建授予您不想在输出容器镜像中保留的信息的访问权限。

构建卷提供仅在构建时需要的敏感信息，如存储库凭据。构建卷与构建输入不同，后者的数据可以保留在输出容器镜像中。

构建卷的挂载点（运行中的构建从中读取数据）在功能上与 pod 卷挂载 类似。

先决条件

- 您已将输入 secret 和配置映射添加到 BuildConfig 对象中。

流程

- 在 **BuildConfig** 对象的 **sourceStrategy** 定义中，将任何构建卷添加到 **volumes** 数组中。例如：

```
spec:
  sourceStrategy:
    volumes:
      - name: secret-mvn (1)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh (2)
        source:
          type: Secret (3)
          secret:
            secretName: my-secret (4)
      - name: settings-mvn (5)
        mounts:
        - destinationPath: /opt/app-root/src/.m2 (6)
```

```
      source:
        type: ConfigMap (7)
        configMap:
          name: my-config (8)
      - name: my-csi-volume (9)
        mounts:
        - destinationPath: /opt/app-root/src/some_path (10)
        source:
          type: CSI (11)
          csi:
            driver: csi.sharedresource.openshift.io (12)
            readOnly: true (13)
            volumeAttributes: (14)
              attribute: value
```

(1) (5) (9) 必需。唯一的名称。

(2) (6) (10) 必需。挂载点的绝对路径。它不能包含 **..** 或 **:** 且不与构建器生成的目的地路径冲突。**/opt/app-root/src** 是许多支持 Red Hat S2I 的镜像的默认主目录。

(3) (7) (11) 必需。源类型，**ConfigMap**、**Secret** 或 **CSI**。

(4) (8) 必需。源的名称。

(12) 必需。提供临时 CSI 卷的驱动程序。

(13) 可选。如果为 true，这指示驱动程序提供只读卷。

(14) 可选。临时 CSI 卷的卷属性。如需支持的属性键和值，请参阅 CSI 驱动程序的文档。

注意

共享资源 CSI 驱动程序作为技术预览提供。

2.5.3. Custom 构建

采用自定义构建策略时，开发人员可以定义负责整个构建过程的特定构建器镜像。通过利用自己的构建器镜像，可以自定义构建流程。

自定义构建器镜像是嵌入了构建过程逻辑的普通容器镜像，例如用于构建 RPM 或基本镜像的逻辑。

自定义构建以级别很高的特权运行，默认情况下不可供用户使用。只有可赋予集群管理权限的用户才应被授予运行自定义构建的权限。

2.5.3.1. 使用 FROM 镜像进行自定义构建

您可以使用 **customStrategy.from** 部分来指示要用于自定义构建的镜像。

流程

- 设置 **customStrategy.from** 部分：

```
strategy:
```

```
customStrategy:
  from:
    kind: "DockerImage"
    name: "openshift/sti-image-builder"
```

2.5.3.2. 在自定义构建中使用 secret

除了可以添加到所有构建类型的源和镜像的 secret 之外，自定义策略还允许向构建器 Pod 添加任意 secret 列表。

流程

- 要将各个 secret 挂载到特定位置，编辑 **策略** YAML 文件的 **secretSource** 和 **mountPath** 字段：

```
strategy:
  customStrategy:
    secrets:
      - secretSource: 1
          name: "secret1"
        mountPath: "/tmp/secret1" 2
      - secretSource:
          name: "secret2"
        mountPath: "/tmp/secret2"
```

1. **secretSource** 是对与构建相同的命名空间中的 secret 的引用。
2. **mountPath** 是自定义构建器中应挂载 secret 的路径。

2.5.3.3. 使用环境变量进行自定义构建

要将环境变量提供给自定义构建过程使用，您可以在构建配置的 **customStrategy** 定义中添加环境变量。

这里定义的环境变量将传递给运行自定义构建的 Pod。

流程

1. 定义在构建期间使用的自定义 HTTP 代理：

```
customStrategy:
...
  env:
    - name: "HTTP_PROXY"
      value: "http://myproxy.net:5187/"
```

2. 要管理构建配置中定义的环境变量，请输入以下命令：

```
$ oc set env <enter_variables>
```

2.5.3.4. 使用自定义构建器镜像

OpenShift Container Platform 的自定义构建策略允许您定义负责整个构建过程的特定构建器镜像。当您需要构建来生成单独的工件，如软件包、JAR、WAR、可安装的 ZIP 或基础镜像时，请使用自定义构建器镜像。

自定义构建器镜像是嵌入构建过程逻辑的普通容器镜像，用于构建工件，如 RPM 或基础容器镜像。

另外，自定义构建器允许实施任何扩展构建过程，如运行单元或集成测试的 CI/CD 流。

2.5.3.4.1. 自定义构建器镜像

在调用时，自定义构建器镜像将接收以下环境变量以及继续进行构建所需要的信息：

表 2.2. 自定义构建器环境变量

变量名称	描述
BUILD	**Build** 对象定义的完整序列化 JSON。如果必须使用特定的 API 版本进行序列化，您可以在构建配置的自定义策略规格中设置 **buildAPIVersion** 参数。
SOURCE_REPOSITORY	包含要构建的源代码的 Git 存储库的 URL。
SOURCE_URI	使用与 **SOURCE_REPOSITORY** 相同的值。可以使用其中任一个。
SOURCE_CONTEXT_DIR	指定要在构建时使用的 Git 存储库的子目录。只有定义后才出现。
SOURCE_REF	要构建的 Git 引用。
ORIGIN_VERSION	创建此构建对象的 OpenShift Container Platform master 的版本。
OUTPUT_REGISTRY	镜像要推送到的容器镜像 registry。
OUTPUT_IMAGE	所构建镜像的容器镜像标签名称。
PUSH_DOCKERCFG_PATH	用于运行 **podman push** 操作的容器 registry 凭证的路径。

2.5.3.4.2. 自定义构建器工作流

虽然自定义构建器镜像作者在定义构建过程时具有很大的灵活性，但构建器镜像仍必须遵循如下必要的步骤，才能在 OpenShift Container Platform 内无缝运行构建：

1. **Build** 对象定义包含有关构建的输入参数的所有必要信息。
2. 运行构建过程。
3. 如果构建生成了镜像，则将其推送到构建的输出位置（若已定义）。可通过环境变量传递其他输出位置。

2.5.4. Pipeline 构建

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

采用 Pipeline 构建策略时，开发人员可以定义 Jenkins 管道，供 Jenkins 管道插件使用。构建可以由 OpenShift Container Platform 启动、监控和管理，其方式与任何其他构建类型相同。

Pipeline 工作流在 **jenkinsfile** 中定义，或直接嵌入在构建配置中，或者在 Git 存储库中提供并由构建配置引用。

2.5.4.1. 了解 OpenShift Container Platform 管道

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

通过管道（pipeline），您可以控制在 OpenShift Container Platform 上构建、部署和推进您的应用程序。通过结合使用 Jenkins Pipeline 构建策略、**jenkinsfile** 和 Jenkins 客户端插件提供的 OpenShift Container Platform 域特定语言（DSL），您可以为任何场景创建高级构建、测试、部署和推进管道。

OpenShift Container Platform Jenkins 同步插件

OpenShift Container Platform Jenkins 同步插件使构建配置和构建对象与 Jenkins 任务和构建保持同步，并提供以下功能：

- Jenkins 中动态作业并运行创建。
- 从镜像流、镜像流标签或配置映射动态创建代理 Pod 模板。
- 注入环境变量。
- OpenShift Container Platform Web 控制台中的管道视觉化。
- 与 Jenkins Git 插件集成，后者将 OpenShift Container Platform 构建的提交信息传递给 Jenkins Git 插件。
- 将 secret 同步到 Jenkins 凭证条目。

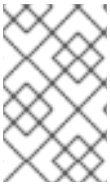
OpenShift Container Platform Jenkins 客户端插件

OpenShift Container Platform Jenkins 客户端插件是一种 Jenkins 插件，旨在提供易读、简洁、全面且流畅的 Jenkins Pipeline 语法，以便与 OpenShift Container Platform API 服务器进行丰富的交互。该插件使用 OpenShift Container Platform 命令行工具 **oc**，此工具必须在执行脚本的节点上可用。

Jenkins 客户端插件必须安装到 Jenkins master上，这样才能在您的应用程序的 **jenkinsfile** 中使用 OpenShift Container Platform DSL。使用 OpenShift Container Platform Jenkins 镜像时，默认安装并启用此插件。

对于项目中的 OpenShift Container Platform 管道，必须使用 Jenkins Pipeline 构建策略。此策略默认使用源存储库根目录下的 **jenkinsfile**，但也提供以下配置选项：

- 构建配置中的内联 **jenkinsfile** 字段。
- 构建配置中的 **jenkinsfilePath** 字段，该字段引用要使用的 **jenkinsfile** 的位置，该位置相对于源 **contextDir**。



注意

可选的 **jenkinsfilePath** 字段指定要使用的文件的名称，其路径相对于源 **contextDir**。如果省略了 **contextDir**，则默认为存储库的根目录。如果省略了 **jenkinsfilePath**，则默认为 **jenkinsfile**。

2.5.4.2. 为管道构建提供 Jenkins 文件

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

jenkinsfile 使用标准的 Groovy 语言语法，允许对应用程序的配置、构建和部署进行精细控制。

您可以通过以下一种方式提供 **jenkinsfile**：

- 位于源代码存储库中的文件。
- 使用 **jenkinsfile** 字段嵌入为构建配置的一部分。

使用第一个选项时，**jenkinsfile** 必须包含在以下位置之一的应用程序源代码存储库中：

- 存储库根目录下名为 **jenkinsfile** 的文件。
- 存储库的源 **contextDir** 的根目录下名为 **jenkinsfile** 的文件。
- 通过 BuildConfig 的 **JenkinsPipelineStrategy** 部分的 **jenkinsfilePath** 字段指定的文件名；若提供，则路径相对于源 **contextDir**，否则默认为存储库的根目录。

jenkinsfile 在 Jenkins 代理 Pod 上运行，如果您打算使用 OpenShift Container Platform DSL，它必须具有 OpenShift Container Platform 客户端二进制文件。

流程

要提供 Jenkins 文件，您可以：

- 在构建配置中嵌入 Jenkins 文件。
- 在构建配置中包含对包含 Jenkins 文件的 Git 存储库的引用。

嵌入式定义

```
kind: "BuildConfig"
apiVersion: "v1"
```

```
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('agent') {
          stage 'build'
          openshiftBuild(buildConfig: 'ruby-sample-build', showBuildLogs: 'true')
          stage 'deploy'
          openshiftDeploy(deploymentConfig: 'frontend')
        }
```

引用 Git 存储库

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  source:
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfilePath: some/repo/dir/filename 1
```

1 可选的 **jenkinsfilePath** 字段指定要使用的文件的名称，其路径相对于源 **contextDir**。如果省略了 **contextDir**，则默认为存储库的根目录。如果省略了 **jenkinsfilePath**，则默认为 **jenkinsfile**。

2.5.4.3. 使用环境变量进行 Pipeline 构建

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

要将环境变量提供给 Pipeline 构建过程使用，您可以在构建配置的 **jenkinsPipelineStrategy** 定义中添加环境变量。

定义后，环境变量将设置为与构建配置关联的任何 Jenkins 任务的参数。

流程

- 要定义在构建期间使用的环境变量，编辑 YAML 文件：

```
jenkinsPipelineStrategy:
...
  env:
```

```
    - name: "FOO"
      value: "BAR"
```

您还可以使用 **oc set env** 命令管理构建配置中定义的环境变量。

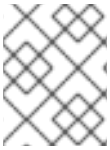
2.5.4.3.1. BuildConfig 环境变量和 Jenkins 任务参数之间的映射

根据对 Pipeline 策略构建配置的更改创建或更新 Jenkins 任务时，构建配置中的任何环境变量都会映射到 Jenkins 任务参数定义，其中 Jenkins 任务参数定义的默认值是关联环境变量的当前值。

在 Jenkins 任务初始创建之后，您仍然可以从 Jenkins 控制台向任务添加其他参数。参数名称与构建配置中的环境变量名称不同。为这些 Jenkins 任务启动构建时，将遵循这些参数。

为 Jenkins 任务启动构建的方式决定了如何设置参数。

- 如果使用 **oc start-build** 启动，则构建配置中的环境变量值是为对应作业实例设置的参数。您在 Jenkins 控制台中对参数默认值所做的更改都将被忽略。构建配置值具有优先权。
- 如果使用 **oc start-build -e** 启动，则 **-e** 选项中指定的环境变量值具有优先权。
  - 如果指定没有列在构建配置中列出的环境变量，它们将添加为 Jenkins 任务参数定义。
  - 您在 Jenkins 控制台中对与环境变量对应的参数所做的更改都将被忽略。构建配置以及您通过 **oc start-build -e** 指定的值具有优先权。
- 如果使用 Jenkins 控制台启动 Jenkins 任务，您可以使用 Jenkins 控制台控制参数的设置，作为启动任务构建的一部分。



注意

建议您在构建配置中指定与作业参数关联的所有可能环境变量。这样做可以减少磁盘 I/O 并提高 Jenkins 处理期间的性能。

2.5.4.4. Pipeline 构建教程

重要

Pipeline 构建策略在 OpenShift Container Platform 4 中弃用。基于 Tekton 的 OpenShift Container Platform Pipelines 中带有等效且改进的功能。

OpenShift Container Platform 上的 Jenkins 镜像被完全支持，用户可以按照 Jenkins 用户文档在作业中定义 **jenkinsfile**，或者将其存储在 Source Control Management 系统中。

本例演示如何创建 OpenShift Container Platform Pipeline，以使用 **nodejs-mongodb.json** 模板构建、部署和验证 **Node.js/MongoDB** 应用程序。

流程

1. 创建 Jenkins master：

```
$ oc project <project_name>
```

选择要使用的项目，或使用 **oc new-project <project_name>** 创建一个新项目。

```
$ oc new-app jenkins-ephemeral 1
```

如果要使用持久性存储，请改用 **jenkins-persistent**。

2. 使用以下内容，创建名为 **nodejs-sample-pipeline.yaml** 的文件：

注意

这将创建一个 **BuildConfig** 对象，它将使用 Jenkins Pipeline 策略来构建、部署和扩展 **Node.js/MongoDB** 示例应用程序。

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "nodejs-sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: <pipeline content from below>
    type: JenkinsPipeline
```

3. 使用 **jenkinsPipelineStrategy** 创建 **BuildConfig** 对象后，通过使用内联 **jenkinsfile** 告知管道做什么：

注意

本例没有为应用程序设置 Git 存储库。

以下 **jenkinsfile** 内容使用 OpenShift Container Platform DSL 以 Groovy 语言编写。在本例中，请使用 YAML Literal Style 在 **BuildConfig** 中包含内联内容，但首选的方法是使用源存储库中的 **jenkinsfile**。

```
def templatePath = 'https://raw.githubusercontent.com/openshift/nodejs-
ex/master/openshift/templates/nodejs-mongodb.json' 1
def templateName = 'nodejs-mongodb-example' 2
pipeline {
  agent {
    node {
      label 'nodejs' 3
    }
  }
  options {
    timeout(time: 20, unit: 'MINUTES') 4
  }
  stages {
    stage('preamble') {
        steps {
            script {
                openshift.withCluster() {
                    openshift.withProject() {
                        echo "Using project: ${openshift.project()}"
                    }
                }
```

```
        }
      }
    }
    stage('cleanup') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              openshift.selector("all", [ template : templateName ]).delete() 5
              if (openshift.selector("secrets", templateName).exists()) { 6
                openshift.selector("secrets", templateName).delete()
              }
            }
          }
        }
      }
    }
    stage('create') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              openshift.newApp(templatePath) 7
            }
          }
        }
      }
    }
    stage('build') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              def builds = openshift.selector("bc", templateName).related('builds')
              timeout(5) { 8
                builds.untilEach(1) {
                  return (it.object().status.phase == "Complete")
                }
              }
            }
          }
        }
      }
    }
    stage('deploy') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              def rm = openshift.selector("dc", templateName).rollout()
              timeout(5) { 9
                openshift.selector("dc", templateName).related('pods').untilEach(1) {
                  return (it.object().status.phase == "Running")
                }
              }
```

```
            }
          }
        }
      }
    }
    stage('tag') {
      steps {
        script {
            openshift.withCluster() {
              openshift.withProject() {
                openshift.tag("${templateName}:latest", "${templateName}-staging:latest") 10
              }
            }
        }
      }
    }
  }
}
```

1 要使用的模板的路径。

1 2 要创建的模板的名称。

3 启动 **node.js** 代理 pod 以针对其运行此构建。

4 为此管道设置 20 分钟超时。

5 使用此模板标签删除所有内容。

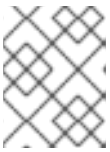
6 使用此模板标签删除任何 secret。

7 从 **templatePath** 创建一个新应用程序。

8 等待最多五分钟以完成构建。

9 等待最多五分钟以完成部署。

10 如果其余部分都成功，则将 **$ {templateName}:latest** 镜像标记为 **$ {templateName}-staging:latest**。stage 环境的管道 BuildConfig 可以监控 **$ {templateName}-staging:latest** 镜像更改，并将它部署到 stage 环境中。



注意

上例使用 declarative pipeline 风格编写，但较旧的 scripted pipeline 风格也受到支持。

4. 在 OpenShift Container Platform 集群中创建管道 **BuildConfig**：

```
$ oc create -f nodejs-sample-pipeline.yaml
```

a. 如果您不想自行创建文件，可以通过运行以下命令来使用 Origin 存储库中的示例：

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/jenkins/pipeline/nodejs-
sample-pipeline.yaml
```

5. 启动管道：

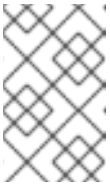
```
$ oc start-build nodejs-sample-pipeline
```

注意

此外，也可以通过 OpenShift Container Platform Web 控制台启动管道，方法是导航到 Builds → Pipeline 部分并点击 **Start Pipeline**，或者访问 Jenkins 控制台，再导航到您创建的管道并点击 **Build Now**。

管道启动之后，您应该看到项目中执行了以下操作：

- 在 Jenkins 服务器上创建了作业实例。
- 如果管道需要，启动一个代理 pod。
- 管道在代理 Pod 上运行，如果不需要代理，则管道在 master 上运行。
  - 将删除之前创建的具有 **template=nodejs-mongodb-example** 标签的所有资源。
  - 从 **nodejs-mongodb-example** 模板创建一个新应用程序及其所有相关资源。
  - 使用 **nodejs-mongodb-example BuildConfig** 启动构建。
    - 管道将等待到构建完成后触发下一阶段。
  - 使用 **nodejs-mongodb-example** 部署配置启动部署。
    - 管道将等待到部署完成后触发下一阶段。
  - 如果构建和部署都成功，则 **nodejs-mongodb-example:latest** 镜像将标记为 **nodejs-mongodb-example:stage**。
- 如果管道需要，则代理 pod 会被删除。



注意

视觉化管道执行的最佳方法是在 OpenShift Container Platform Web 控制台中查看它。您可以通过登录 Web 控制台并导航到 Builds → Pipelines 来查看管道。

2.5.5. 使用 web 控制台添加 secret

您可以在构建配置中添加 secret，以便它可以访问私有存储库。

流程

将 secret 添加到构建配置中，以便它可以从 OpenShift Container Platform Web 控制台访问私有存储库：

1. 创建一个新的 OpenShift Container Platform 项目。

2. 创建一个包含用于访问私有源代码存储库的凭证的 secret。
3. 创建构建配置。
4. 在构建配置编辑器页面上或在 Web 控制台的 **create app from builder image** 页面中，设置 **Source Secret**。
5. 点击 **Save**。

2.5.6. 启用拉取 (pull) 和推送 (push)

您可以通过在构建配置中设置 pull secret 来启用拉取到私有 registry，也可以通过设置 push secret 来启用推送。

流程

启用拉取到私有 registry：

- 在构建配置中设置 pull secret。

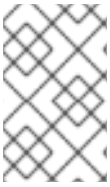
启用推送：

- 在构建配置中设置 push secret。

2.6. 使用 BUILDAH 自定义镜像构建

在 OpenShift Container Platform 4.10 中，主机节点上没有 docker socket。这意味着，不能保证自定义构建的 *mount docker socket* 选项会提供可在自定义构建镜像中使用的可访问 docker socket。

如果您需要此功能来构建和推送镜像，请将 Buildah 工具添加到自定义构建镜像中，并在自定义构建逻辑中使用它来构建并推送镜像。以下是如何使用 Buildah 运行自定义构建的示例。



注意

使用自定义构建策略需要普通用户默认情况下不具备的权限，因为它允许用户在集群上运行的特权容器内执行任意代码。此级别的访问权限可被用来进行可能对集群造成损害的操作，因此应仅授权给信任的用户。

2.6.1. 先决条件

- 查看如何授予自定义构建权限。

2.6.2. 创建自定义构建工件

您必须创建要用作自定义构建镜像的镜像。

流程

1. 从空目录着手，使用以下内容创建名为 **Dockerfile** 的文件：

```
FROM registry.redhat.io/rhel8/buildah
# In this example, `/tmp/build` contains the inputs that build when this
# custom builder image is run. Normally the custom builder image fetches
# this content from some location at build time, by using git clone as an example.
ADD dockerfile.sample /tmp/input/Dockerfile
```

```
ADD build.sh /usr/bin
RUN chmod a+x /usr/bin/build.sh
# /usr/bin/build.sh contains the actual custom build logic that will be run when
# this custom builder image is run.
ENTRYPOINT ["/usr/bin/build.sh"]
```

2. 在同一目录中，创建名为 **dockerfile.sample** 的文件。此文件将包含在自定义构建镜像中，并且定义将由自定义构建生成的镜像：

```
FROM registry.access.redhat.com/ubi8/ubi
RUN touch /tmp/build
```

3. 在同一目录中，创建名为 **build.sh** 的文件。此文件包含自定义生成运行时将要执行的逻辑：

```
#!/bin/sh
# Note that in this case the build inputs are part of the custom builder image, but normally this
# is retrieved from an external source.
cd /tmp/input
# OUTPUT_REGISTRY and OUTPUT_IMAGE are env variables provided by the custom
# build framework
TAG="${OUTPUT_REGISTRY}/${OUTPUT_IMAGE}"


# performs the build of the new image defined by dockerfile.sample
buildah --storage-driver vfs bud --isolation chroot -t ${TAG} .


# buildah requires a slight modification to the push secret provided by the service
# account to use it for pushing the image
cp /var/run/secrets/openshift.io/push/.dockercfg /tmp
(echo "{ \"auths\": " ; cat /var/run/secrets/openshift.io/push/.dockercfg ; echo "}") >
/tmp/.dockercfg


# push the new image to the target for the build
buildah --storage-driver vfs push --tls-verify=false --authfile /tmp/.dockercfg ${TAG}
```

2.6.3. 构建自定义构建器镜像

您可以使用 OpenShift Container Platform 构建和推送要在 Custom 策略中使用的自定义构建器镜像。

先决条件

- 定义要用于创建新的自定义构建器镜像的所有输入。

流程

1. 定义要用于构建自定义构建器镜像的 **BuildConfig** 对象：

```
$ oc new-build --binary --strategy=docker --name custom-builder-image
```

2. 从您在其中创建自定义构建镜像的目录中，运行构建：

```
$ oc start-build custom-builder-image --from-dir . -F
```

- 构建完成后，新自定义构建器镜像将在名为 **custom-builder-image:latest** 的镜像流标签中的项目内可用。

2.6.4. 使用自定义构建器镜像

您可以定义一个 **BuildConfig** 对象，它将结合使用 Custom 策略与自定义构建器镜像来执行您的自定义构建逻辑。

先决条件

- 为新自定义构建器镜像定义所有必要的输入。
- 构建您的自定义构建器镜像。

流程

1. 创建名为 **buildconfig.yaml** 的文件。此文件定义要在项目中创建并执行的 **BuildConfig** 对象：

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: sample-custom-build
  labels:
    name: sample-custom-build
  annotations:
    template.alpha.openshift.io/wait-for-ready: 'true'
spec:
  strategy:
    type: Custom
    customStrategy:
      forcePull: true
      from:
        kind: ImageStreamTag
        name: custom-builder-image:latest
        namespace: <yourproject> (1)
  output:
    to:
      kind: ImageStreamTag
      name: sample-custom:latest
```

(1) 指定项目的名称。

2. 创建 **BuildConfig**:

```
$ oc create -f buildconfig.yaml
```

3. 创建名为 **imagestream.yaml** 的文件。此文件定义构建要将镜像推送到的镜像流：

```
kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: sample-custom
spec: {}
```

4. 创建镜像流：

```
$ oc create -f imagestream.yaml
```

5. 运行自定义构建：

```
$ oc start-build sample-custom-build -F
```

构建运行时，它会启动一个 Pod 来运行之前构建的自定义构建器镜像。该 Pod 将运行定义为自定义构建器镜像入口点的 **build.sh** 逻辑。**build.sh** 逻辑调用 Buildah 来构建自定义构建器镜像中嵌入的 **dockerfile.sample**，然后使用 Buildah 将新镜像推送到 **sample-custom 镜像流**。

2.7. 执行和配置基本构建

以下小节提供了有关基本构建操作的说明，包括启动和取消构建、编辑 **BuildConfig**、删除 **BuildConfig**、查看构建详情以及访问构建日志。

2.7.1. 启动构建

您可以从当前项目中的现有构建配置手动启动新构建。

流程

要手动启动构建，请输入以下命令：

```
$ oc start-build <buildconfig_name>
```

2.7.1.1. 重新运行构建

您可以使用 **--from-build** 标志，手动重新运行构建。

流程

- 要手动重新运行构建，请输入以下命令：

```
$ oc start-build --from-build=<build_name>
```

2.7.1.2. 流传输构建日志

您可以指定 **--follow** 标志，在 **stdout** 中输出构建日志。

流程

- 要在 **stdout** 中手动输出构建日志，请输入以下命令：

```
$ oc start-build <buildconfig_name> --follow
```

2.7.1.3. 在启动构建时设置环境变量

您可以指定 **--env** 标志，为构建设置任何所需的环境变量。

流程

- 要指定所需的环境变量，请输入以下命令：

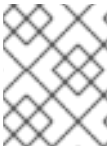
```
$ oc start-build <buildconfig_name> --env=<key>=<value>
```

2.7.1.4. 使用源启动构建

您可以通过直接推送源来启动构建，而不依赖于 Git 源拉取或构建的 Dockerfile；源可以是 Git 或 SVN 工作目录的内容、您想要部署的一组预构建二进制工件，或者单个文件。这可以通过为 **start-build** 命令指定以下选项之一来完成：

选项	描述
--from-dir=<directory>	指定将要存档并用作构建的二进制输入的目录。
--from-file=<file>	指定将成为构建源中唯一文件的单个文件。该文件放在空目录的根目录中，其文件名与提供的原始文件相同。
--from-repo=<local_source_repo>	指定用作构建二进制输入的本地存储库的路径。添加 **--commit** 选项以控制要用于构建的分支、标签或提交。

将任何这些选项直接传递给构建时，内容将流传输到构建中并覆盖当前的构建源设置。



注意

从二进制输入触发的构建不会在服务器上保留源，因此基础镜像更改触发的重新构建将使用构建配置中指定的源。

流程

- 使用以下命令从源启动构建，以将本地 Git 存储库的内容作为标签 **v2** 的存档发送：

```
$ oc start-build hello-world --from-repo=../hello-world --commit=v2
```

2.7.2. 取消构建

您可以使用 Web 控制台或通过以下 CLI 命令来取消构建。

流程

- 要手动取消构建，请输入以下命令：

```
$ oc cancel-build <build_name>
```

2.7.2.1. 取消多个构建

您可以使用以下 CLI 命令取消多个构建。

流程

- 要手动取消多个构建，请输入以下命令：

```
$ oc cancel-build <build1_name> <build2_name> <build3_name>
```

2.7.2.2. 取消所有构建

您可以使用以下 CLI 命令取消构建配置中的所有构建。

流程

- 要取消所有构建，请输入以下命令：

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.2.3. 取消给定状态下的所有构建

您可以取消给定状态下的所有构建，如 **new** 或 **pending** 状态，同时忽略其他状态下的构建。

流程

- 要取消给定状态下的所有内容，请输入以下命令：

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.3. 编辑 BuildConfig

要编辑构建配置，您可以使用 Developer 视角的 **Builds** 视图中的 **Edit BuildConfig** 选项。

您可以使用以下任一视图编辑 **BuildConfig** ：

- **Form 视图** 允许您使用标准表单字段和复选框编辑 **BuildConfig**。
- **YAML 视图** 允许您编辑 **BuildConfig**，完全控制操作。

您可以在 **Form view** 和 **YAML 视图**间切换，而不丢失任何数据。**Form 视图**中的数据传输到 **YAML 视图**，反之亦然。

流程

1. 在 **Developer 视角** 的 **Builds** 视图中，点击菜单  来查看 **Edit BuildConfig** 选项。
2. 点击 **Edit BuildConfig** 以查看 **Form view** 选项。
3. 在 **Git** 部分中，输入您要用来创建应用程序的代码库的 Git 存储库 URL。这个 URL 随后会被验证。
   - 可选：点击 **Show Advanced Git Options** 来添加详情，例如：
     - **Git Reference**，用于指定包含您要用来构建应用程序的代码的分支、标签或提交。
     - **Context Dir**，用于指定包含您要用来构建应用程序的代码的子目录。

- Source Secret，创建一个具有用来从私有存储库拉取源代码的凭证的 Secret Name。

4. 在 Build from 部分中，选择您要从中构建的选项。您可以使用以下选项：

- **镜像流标签** 引用给定镜像流和标签的镜像。输入您要从构建并推送到的位置的项目、镜像流和标签。
- **镜像流镜像** 引用给定镜像流和镜像名称的镜像。输入您要从中构建的镜像流镜像。另外，进入要推送到的项目、镜像流和标签。
- **Docker 镜像** ：通过 Docker 镜像存储库引用 Docker 镜像。您还需要进入项目、镜像流和标签，以引用您要推送到的位置。

5. 可选：在 Environment Variables 部分中，使用 Name 和 Value 字段添加与项目关联的环境变量。要添加更多环境变量，请使用 Add Value 或 Add from ConfigMap 和 Secret。

6. 可选： 要进一步自定义应用程序，请使用以下高级选项：

Trigger

构建器镜像更改时触发新镜像构建。点 Add Trigger 并选择 Type 和 Secret 来添加更多触发器。

Secrets

为应用添加 secret。点 Add secret 并选择 Secret 和 Mountpoint 来添加更多 secret。

策略

单击 Run policy 以选择构建运行策略。所选策略决定从构建配置创建的构建必须运行的顺序。

Hook

选择择 Run build hooks after image is built，以在构建结束时运行命令并验证镜像。添加 Hook 类型、命令 和 参数，以附加到 命令。

7. 单击 Save 以保存 BuildConfig。

2.7.4. 删除 BuildConfig

您可以使用以下命令来删除 BuildConfig。

流程

- 要删除 BuildConfig，请输入以下命令：

```
$ oc delete bc <BuildConfigName>
```

这也会删除从此 BuildConfig 实例化的所有构建。

- 要删除 BuildConfig 并保留从 BuildConfig 中初始化的构建，在输入以下命令时指定 --cascade=false 标志：

```
$ oc delete --cascade=false bc <BuildConfigName>
```

2.7.5. 查看构建详情

您可以使用 Web 控制台或 oc describe CLI 命令查看构建详情。

这会显示，包括：

- 构建源。
- 构建策略。
- 输出目的地。
- 目标 registry 中的镜像摘要。
- 构建的创建方式。

如果构建采用 **Docker** 或 **Source** 策略，则 **oc describe** 输出还包括用于构建的源修订的相关信息，包括提交 ID、作者、提交者和消息等。

流程

- 要查看构建详情，请输入以下命令：

```
$ oc describe build <build_name>
```

2.7.6. 访问构建日志

您可以使用 Web 控制台或 CLI 访问构建日志。

流程

- 要直接使用构建来流传输日志，请输入以下命令：

```
$ oc describe build <build_name>
```

2.7.6.1. 访问 BuildConfig 日志

您可以使用 Web 控制台或 CLI 访问 **BuildConfig** 日志。

流程

- 要输出 **BuildConfig** 的最新构建的日志，请输入以下命令：

```
$ oc logs -f bc/<buildconfig_name>
```

2.7.6.2. 访问给定版本构建的 BuildConfig 日志

您可以使用 Web 控制台或 CLI 访问 **BuildConfig** 的给定版本构建的日志。

流程

- 要输出 **BuildConfig** 的给定版本构建的日志，请输入以下命令：

```
$ oc logs --version=<number> bc/<buildconfig_name>
```

2.7.6.3. 启用日志详细程度

您可以传递 **BUILD_LOGLEVEL** 环境变量作为 **BuildConfig** 中 **sourceStrategy** 或 **dockerStrategy** 的一部分，来实现更为详细的输出。

注意

管理员可以通过配置 **env/BUILD_LOGLEVEL**，为整个 OpenShift Container Platform 实例设置默认的构建详细程度。此默认值可以通过在给定的 **BuildConfig** 中指定 **BUILD_LOGLEVEL** 来覆盖。您可以通过将 **--build-loglevel** 传递给 **oc start-build**，在命令行中为非二进制构建指定优先级更高的覆盖。

源构建的可用日志级别如下：

0 级	生成运行 **assemble** 脚本的容器的输出，以及所有遇到的错误。这是默认值。
1 级	生成有关已执行进程的基本信息。
2 级	生成有关已执行进程的非常详细的信息。
3 级	生成有关已执行进程的非常详细的信息，以及存档内容的列表。
4 级	目前生成与 3 级相同的信息。
5 级	生成以上级别中包括的所有内容，另外还提供 Docker 推送消息。

流程

- 要启用更为详细的输出，请传递 **BUILD_LOGLEVEL** 环境变量作为 **BuildConfig** 中 **sourceStrategy** 或 **dockerStrategy** 的一部分：

```
sourceStrategy:
...
  env:
    - name: "BUILD_LOGLEVEL"
      value: "2" 1
```

1 将此值调整为所需的日志级别。

2.8. 触发和修改构建

以下小节概述了如何使用构建 hook 触发构建和修改构建。

2.8.1. 构建触发器

在定义 **BuildConfig** 时，您可以定义触发器来控制应该运行 **BuildConfig** 的环境。可用的构建触发器如下：

- Webhook
- 镜像更改

- 配置更改

2.8.1.1. Webhook 触发器

Webhook 触发器通过发送请求到 OpenShift Container Platform API 端点来触发新构建。您可以使用 GitHub、GitLab、Bitbucket 或通用 Webhook 来定义这些触发器。

目前，OpenShift Container Platform Webhook 仅支持各种基于 Git 的源代码管理系统 (SCM) 的推送事件的类同版本。所有其他事件类型都会忽略。

处理推送事件时，OpenShift Container Platform control plane 主机确认事件内的分支引用是否与对应 **BuildConfig** 中的分支引用匹配。如果匹配，它会检查 OpenShift Container Platform 构建的 Webhook 事件中记录的确切提交引用。如果不匹配，则不触发构建。

注意

oc new-app 和 **oc new-build** 会自动创建 GitHub 和通用 Webhook 触发器，但其他所需的 Webhook 触发器都必须手动添加。您可以通过设置触发器来手动添加触发器。

对于所有 Webhook，您必须使用名为 **WebHookSecretKey** 的键定义 secret，并且其值是调用 Webhook 时要提供的值。然后，Webhook 定义必须引用该 secret。secret可确保 URL 的唯一性，防止他人触发构建。键的值将与 Webhook 调用期间提供的 secret 进行比较。

例如，此处的 GitHub Webhook 具有对名为 **mysecret** 的 secret 的引用：

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```

该 secret 的定义如下。注意 secret 的值采用 base64 编码，如 **Secret** 对象的 **data** 字段所要求。

```
- kind: Secret
  apiVersion: v1
  metadata:
    name: mysecret
    creationTimestamp:
  data:
    WebHookSecretKey: c2VjcmV0dmFsdWUx
```

2.8.1.1.1. 使用 GitHub Webhook

当存储库更新时，GitHub Webhook 处理 GitHub 发出的调用。在定义触发器时，您必须指定一个 secret，它将是您在配置 Webhook 时提供给 GitHub 的 URL 的一部分。

GitHub Webhook 定义示例：

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```

注意

Webhook 触发器配置中使用的 secret 与在 GitHub UI 中配置 Webhook 时遇到的 **secret** 字段不同。前者使 Webhook URL 唯一且难以预测，后者是一个可选的字符串字段，用于创建正文的 HMAC 十六进制摘要，作为 **X-Hub-Signature** 标头来发送。

oc describe 命令将有效负载 URL 返回为 GitHub Webhook URL（请参阅"显示 Webhook URL"），其结构如下：

输出示例

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

先决条件

- 从 GitHub 存储库创建 **BuildConfig**。

流程

1. 配置 GitHub Webhook：
   a. 从 GitHub 存储库创建 **BuildConfig** 后，运行以下命令：

   ```
   $ oc describe bc/<name-of-your-BuildConfig>
   ```

   这会生成一个 Webhook GitHub URL，如下所示：

   输出示例

   ```
   <https://api.starter-us-east-1.openshift.com:443/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
   ```

   b. 从 GitHub Web 控制台将此 URL 剪切并粘贴到 GitHub 中。

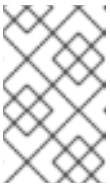
   c. 在 GitHub 存储库中，从 **Settings → Webhooks** 中选择 **Add Webhook**。

   d. 将 URL 输出粘贴到 **Payload URL** 字段。

   e. 将 **Content Type** 从 GitHub 默认的 **application/x-www-form-urlencoded** 更改为 **application/json**。

   f. 点击 **Add webhook**。
   您应该看到一条来自 GitHub 的消息，说明您的 Webhook 已配置成功。

   现在，每当您将更改推送到 GitHub 存储库时，新构建会自动启动，成功构建后也会启动新部署。



注意

Gogs 支持与 GitHub 相同的 Webhook 有效负载格式。因此，如果您使用的是 Gogs 服务器，也可以在 **BuildConfig** 中定义 GitHub Webhook 触发器，并由 Gogs 服务器触发它。

2. 提供含有有效 JSON 内容的文件后，如 **payload.json**，您可以使用 **curl** 手动触发 Webhook：

```
$ curl -H "X-GitHub-Event: push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

只有在 API 服务器没有适当签名的证书时，才需要 **-k** 参数。

注意

只有 GitHub Webhook 事件的 **ref** 值与 **BuildConfig** 资源中的 **source.git** 字段中指定的 **ref** 值匹配时，才会触发构建。

其他资源

- Gogs

2.8.1.1.2. 使用 GitLab Webhook

当存储库更新时，GitLab Webhook 处理 GitLab 发出的调用。与 GitHub 触发器一样，您必须指定一个 secret。以下示例是 **BuildConfig** 中的触发器定义 YAML：

```
type: "GitLab"
gitlab:
  secretReference:
    name: "mysecret"
```

oc describe 命令将有效负载 URL 返回为 GitLab Webhook URL，其结构如下：

输出示例

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

流程

1. 配置 GitLab Webhook：
   a. 描述 **BuildConfig** 以获取 Webhook URL：

```
$ oc describe bc <name>
```

   b. 复制 Webhook URL，将 **<secret>** 替换为您的 secret 值。
   c. 按照 GitLab 设置说明，将 Webhook URL 粘贴到 GitLab 存储库设置中。
2. 提供含有有效 JSON 内容的文件后，如 **payload.json**，您可以使用 **curl** 手动触发 Webhook：

```
$ curl -H "X-GitLab-Event: Push Hook" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

只有在 API 服务器没有适当签名的证书时，才需要 **-k** 参数。

2.8.1.1.3. 使用 Bitbucket Webhook

当存储库更新时，Bitbucket Webhook 处理 Bitbucket 发出的调用。与前面的触发器类似，您必须指定一个 secret。以下示例是 **BuildConfig** 中的触发器定义 YAML：

```
type: "Bitbucket"
bitbucket:
  secretReference:
    name: "mysecret"
```

oc describe 命令将有效负载 URL 返回为 Bitbucket Webhook URL，其结构如下：

输出示例

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

流程

1. 配置 Bitbucket Webhook：
   a. 描述 BuildConfig 以获取 Webhook URL：

      ```
      $ oc describe bc <name>
      ```

   b. 复制 Webhook URL，将 **<secret>** 替换为您的 secret 值。
   c. 按照 Bitbucket 设置说明，将 Webhook URL 粘贴到 Bitbucket 存储库设置中。
2. 提供含有有效 JSON 内容的文件后，如 **payload.json**，您可以使用 **curl** 手动触发 Webhook：

   ```
   $ curl -H "X-Event-Key: repo:push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
   ```

   只有在 API 服务器没有适当签名的证书时，才需要 **-k** 参数。

2.8.1.1.4. 使用通用 Webhook

通用 Webhook 可从能够发出 Web 请求的任何系统调用。与其他 Webhook一样，您必须指定一个 secret，该 secret 将成为调用者必须用于触发构建的 URL 的一部分。secret可确保 URL 的唯一性，防止他人触发构建。如下是 **BuildConfig** 中的示例触发器定义 YAML：

```
type: "Generic"
generic:
  secretReference:
    name: "mysecret"
  allowEnv: true (1)
```

(1) 设置为 **true**，以允许通用 Webhook 传入环境变量。

流程

1. 要设置调用者，请为调用系统提供构建的通用 Webhook 端点的 URL：

   输出示例

   ```
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/generic
   ```

   调用者必须以 **POST** 操作形式调用 Webhook。

2. 要手动调用 Webhook，您可以使用 **curl**：

   ```
   $ curl -X POST -k https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/generic
   ```

   HTTP 操作动词必须设置为 **POST**。指定了不安全 **-k** 标志以忽略证书验证。如果集群拥有正确签名的证书，则不需要此第二个标志。

   端点可以接受具有以下格式的可选有效负载：

   ```
   git:
     uri: "<url to git repository>"
     ref: "<optional git reference>"
     commit: "<commit hash identifying a specific git commit>"
     author:
       name: "<author name>"
       email: "<author e-mail>"
     committer:
       name: "<committer name>"
       email: "<committer e-mail>"
     message: "<commit message>"
   env: (1)
     - name: "<variable name>"
       value: "<variable value>"
   ```

   (1) 与 **BuildConfig** 环境变量类似，此处定义的环境变量也可供您的构建使用。如果这些变量与 **BuildConfig** 环境变量发生冲突，则以这些变量为准。默认情况下，Webhook 传递的环境变量将被忽略。在 Webhook 定义上将 **allowEnv** 字段设为 **true** 即可启用此行为。

3. 要使用 **curl** 传递此有效负载，请在名为 **payload_file.yaml** 的文件中进行定义，再运行以下命令：

   ```
   $ curl -H "Content-Type: application/yaml" --data-binary @payload_file.yaml -X POST -k https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/generic
   ```

   参数与前一个示例相同，但添加了标头和 payload。**-H** 参数将 **Content-Type** 标头设置为 **application/yaml** 或 **application/json**，具体取决于您的 payload 格式。**--data-binary** 参数用于通过 **POST** 请求发送带有换行符的二进制 payload。

注意

即使出示了无效的请求 payload（例如，无效的内容类型，或者无法解析或无效的内容等），OpenShift Container Platform 也允许通用 Webhook 触发构建。保留此行为是为了向后兼容。如果出示无效的请求 payload，OpenShift Container Platform 将以 JSON 格式返回警告，作为其 **HTTP 200 OK** 响应的一部分。

2.8.1.1.5. 显示 Webhook URL

您可以使用以下命令来显示与构建配置关联的 Webhook URL。如果命令不显示任何 Webhook URL，则没有为该构建配置定义任何 Webhook 触发器。

流程

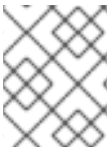
- 显示与 **BuildConfig** 关联的任何 Webhook URL：

```
$ oc describe bc <name>
```

2.8.1.2. 使用镜像更改触发器

作为开发人员，您可以将构建配置为在每次基础镜像更改时自动运行。

当上游镜像有新版本可用时，您可以使用镜像更改触发器自动调用构建。例如，如果构建基于 RHEL 镜像，您可以触发该构建在 RHEL 镜像更改时运行。因此，应用程序镜像始终在最新的 RHEL 基础镜像上运行。



注意

指向 v1 容器 registry 中的容器镜像的镜像流仅在镜像流标签可用时触发一次构建，后续镜像更新时则不会触发。这是因为 v1 容器 registry 中缺少可唯一标识的镜像。

流程

1. 定义指向您要用作触发器的上游镜像的 **ImageStream** ：

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby-20-centos7"
```

这将定义绑定到位于 ***<system-registry>*/*<namespace>*/ruby-20-centos7** 的容器镜像存储库的镜像流。**<system-registry>** 定义为 OpenShift Container Platform 中运行的名为 **docker-registry** 的服务。

2. 如果镜像流是构建的基础镜像，请将构建策略中的 **from** 字段设置为指向 **ImageStream**：

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
```

在这种情形中，**sourceStrategy** 定义将消耗此命名空间中名为 **ruby-20-centos7** 的镜像流的 **latest** 标签。

3. 使用指向 **ImageStreams** 的一个或多个触发器定义构建：

```
type: "ImageChange" 1
imageChange: {}
type: "ImageChange" 2
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
```

1 监控构建策略的 **from** 字段中定义的 **ImageStream** 和 **Tag** 的镜像更改触发器。此处的 **imageChange** 对象必须留空。

2 监控任意镜像流的镜像更改触发器。此时 **imageChange** 部分必须包含一个 **from** 字段，以引用要监控的 **ImageStreamTag**。

将镜像更改触发器用于策略镜像流时，生成的构建会获得一个不可变 docker 标签，指向与该标签对应的最新镜像。在执行构建时，策略会使用此新镜像引用。

对于不引用策略镜像流的其他镜像更改触发器，系统会启动新构建，但不会使用唯一镜像引用来更新构建策略。

由于此示例具有策略的镜像更改触发器，因此生成的构建将是：

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "172.30.17.3:5001/mynamespace/ruby-20-centos7:<immutableid>"
```

这将确保触发的构建使用刚才推送到存储库的新镜像，并且可以使用相同的输入随时重新运行构建。

您可以暂停镜像更改触发器，以便在构建开始之前对引用的镜像流进行多次更改。在将 **ImageChangeTrigger** 添加到 **BuildConfig** 时，您也可以将 **paused** 属性设为 true，以避免立即触发构建。

```
type: "ImageChange"
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
  paused: true
```

除了设置适用于所有 **Strategy** 类型的镜像字段外，自定义构建还需要检查 **OPENSHIFT_CUSTOM_BUILD_BASE_IMAGE** 环境变量。如果不存在，则使用不可变镜像引用来创建它。如果存在，则使用不可变镜像引用进行更新。

如果因为 Webhook 触发器或手动请求而触发构建，则创建的构建将使用从 **Strategy** 引用的 **ImageStream** 解析而来的 **<immutableid>**。这将确保使用一致的镜像标签来执行构建，以方便再生。

其他资源

- v1 容器 registry

2.8.1.3. 识别构建的镜像更改触发器

作为开发人员，如果您有镜像更改触发器，您可以识别启动了上一次构建的镜像更改。这对于调试或故障排除构建非常有用。

BuildConfig示例

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: bc-ict-example
  namespace: bc-ict-example-namespace
spec:

# ...

  triggers:
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input:latest
        namespace: bc-ict-example-namespace
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input2:latest
        namespace: bc-ict-example-namespace
    type: ImageChange
status:
  imageChangeTriggers:
  - from:
      name: input:latest
      namespace: bc-ict-example-namespace
    lastTriggerTime: "2021-06-30T13:47:53Z"
    lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-
namespace/input@sha256:0f88ffbeb9d25525720bfa3524cb1bf0908b7f791057cf1acfae917b11266a69

  - from:
      name: input2:latest
      namespace: bc-ict-example-namespace
    lastTriggeredImageID:  image-registry.openshift-image-registry.svc:5000/bc-ict-example-
namespace/input2@sha256:0f88ffbeb9d25525720bfa3524cb2ce0908b7f791057cf1acfae917b11266a6
9

  lastVersion: 1
```

注意

本例省略了与镜像更改触发器无关的元素。

先决条件

- 您已配置了多个镜像更改触发器。这些触发器已触发一个或多个构建。

流程

1. 在 **buildConfig.status.imageChangeTriggers** 中，标识具有最新时间戳的 **lastTriggerTime**。这个 **ImageChangeTriggerStatus**

> Then you use the `name` and `namespace` from that build to find the corresponding image change trigger in `buildConfig.spec.triggers`.

2. 在 **UnderimageChangeTriggers** 下，比较时间戳以标识最新的

镜像更改触发器

在构建配置中，**buildConfig.spec.triggers** 是构建触发器策略 **BuildTriggerPolicy** 的数组。

每个 **BuildTriggerPolicy** 都有 **type** 字段和指针字段。每个指针字段对应于 **type** 字段允许的值之一。因此，您只能将 **BuildTriggerPolicy** 设置为一个指针字段。

对于镜像更改触发器，**type** 的值为 **ImageChange**。然后，**imageChange** 字段是指向 **ImageChangeTrigger** 对象的指针，其具有以下字段：

- **lastTriggeredImageID**：此字段在 OpenShift Container Platform 4.8 中已弃用，并将在以后的发行版本中被忽略。它包含从此 **BuildConfig** 触发最后一次构建时的 **ImageStreamTag** 的已解析镜像引用。
- **paused**：您可以使用此字段（示例中未显示）暂时禁用此特定镜像更改触发器。
- **from**：您使用此字段引用驱动此镜像更改触发器的 **ImageStreamTag**。其类型是核心 Kubernetes 类型 **OwnerReference**。

from 字段有以下备注字段： **kind: 对于镜像更改触发器，唯一支持的值是 ImageStreamTag。** **namespace**：您可以使用此字段指定 **ImageStreamTag** 的命名空间。** **name** ：您可以使用此字段指定 **ImageStreamTag**。

镜像更改触发器状态

在构建配置中，**buildConfig.status.imageChangeTriggers** 是 **ImageChangeTriggerStatus** 元素的数组。每个 **ImageChangeTriggerStatus** 元素都包含上例中所示的 **from**、**lastTriggeredImageID** 和 **lastTriggerTime** 元素。

具有最新 **lastTriggerTime** 的 **ImageChangeTriggerStatus** 触发了最新的构建。您可以使用其 **name** 和 **namespace** 来识别触发构建的 **buildConfig.spec.triggers** 中的镜像更改触发器。

带有最新时间戳的 **lastTriggerTime** 表示最后一个构建的 **ImageChangeTriggerStatus**。此 **ImageChangeTriggerStatus** 的**name** 和 **namespace** 与触发构建的 **buildConfig.spec.triggers** 中的镜像更改触发器相同。

其他资源

- v1 容器 registry

2.8.1.4. 配置更改触发器

通过配置更改触发器，您可以在创建新 **BuildConfig** 时立即自动调用构建。

如下是 **BuildConfig** 中的示例触发器定义 YAML：

```
type: "ConfigChange"
```

注意

配置更改触发器目前仅在创建新 **BuildConfig** 时运作。在未来的版本中，配置更改触发器也可以在每当 **BuildConfig** 更新时启动构建。

2.8.1.4.1. 手动设置触发器

您可以使用 **oc set triggers** 在构建配置中添加和移除触发器。

流程

- 要在构建配置上设置 GitHub Webhook 触发器，请使用：

```
$ oc set triggers bc <name> --from-github
```

- 要设置镜像更改触发器，请使用：

```
$ oc set triggers bc <name> --from-image='<image>'
```

- 要移除触发器，请添加 **--remove**：

```
$ oc set triggers bc <name> --from-bitbucket --remove
```

注意

如果 Webhook 触发器已存在，再次添加它会重新生成 Webhook secret。

如需更多信息，请查阅帮助文档

```
$ oc set triggers --help
```

2.8.2. 构建 hook

通过构建 hook，可以将行为注入到构建过程中。

BuildConfig 对象的 **postCommit** 字段在运行构建输出镜像的临时容器内执行命令。Hook 的执行时间是紧接在提交镜像的最后一层后，并且在镜像推送到 registry 之前。

当前工作目录设置为镜像的 **WORKDIR**，即容器镜像的默认工作目录。对于大多数镜像，这是源代码所处的位置。

如果脚本或命令返回非零退出代码，或者启动临时容器失败，则 hook 将失败。当 hook 失败时，它会将构建标记为失败，并且镜像也不会推送到 registry。可以通过查看构建日志来检查失败的原因。

构建 hook 可用于运行单元测试，以在构建标记为完成并在 registry 中提供镜像之前验证镜像。如果所有测试都通过并且测试运行器返回退出代码 **0**，则构建标记为成功。如果有任何测试失败，则构建标记为失败。在所有情况下，构建日志将包含测试运行器的输出，这可用于识别失败的测试。

postCommit hook 不仅限于运行测试，也可用于运行其他命令。由于它在临时容器内运行，因此 hook 所做的更改不会持久存在；也就是说，hook 执行无法对最终镜像造成影响。除了其他用途外，也可借助此行为来安装和使用会自动丢弃并且不出现在最终镜像中的测试依赖项。

2.8.2.1. 配置提交后构建 hook

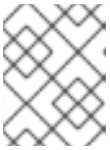
配置提交后构建 hook 的方法有多种。以下示例中所有形式具有同等作用，也都执行 **bundle exec rake test --verbose**。

流程

- Shell 脚本：

```
postCommit:
  script: "bundle exec rake test --verbose"
```

script 值是通过 **/bin/sh -ic** 执行的 shell 脚本。当 shell 脚本适合执行构建 hook 时可使用此选项。例如，用于运行前文所述的单元测试。若要控制镜像入口点，或者如果镜像没有 **/bin/sh**，可使用 **command** 和/或 **args**。



注意

引入的额外 **-i** 标志用于改进搭配 CentOS 和 RHEL 镜像时的体验，未来的发行版中可能会剔除。

- 命令作为镜像入口点：

```
postCommit:
  command: ["/bin/bash", "-c", "bundle exec rake test --verbose"]
```

在这种形式中，**command** 是要运行的命令，它会覆盖 exec 形式中的镜像入口点，如 Dockerfile 引用中所述。如果镜像没有 **/bin/sh**，或者您不想使用 shell，则需要这样做。在所有其他情形中，使用 **script** 可能更为方便。

- 命令带有参数：

```
postCommit:
  command: ["bundle", "exec", "rake", "test"]
  args: ["--verbose"]
```

这种形式相当于将参数附加到 **command**。

注意

同时提供 **script** 和 **command** 会产生无效的构建 hook。

2.8.2.2. 使用 CLI 设置提交后构建 hook

oc set build-hook 命令可用于为构建配置设置构建 hook。

流程

1. 将命令设置为提交后构建 hook：

```
$ oc set build-hook bc/mybc \
    --post-commit \
    --command \
    -- bundle exec rake test --verbose
```

2. 将脚本设置为提交后构建 hook：

```
$ oc set build-hook bc/mybc --post-commit --script="bundle exec rake test --verbose"
```

2.9. 执行高级构建

以下小节提供了有关高级构建操作的说明，包括设置构建资源和最长持续时间、将构建分配给节点、串联构建、修剪构建，以及构建运行策略。

2.9.1. 设置构建资源

默认情况下，构建由 Pod 使用未绑定的资源（如内存和 CPU）来完成。这些资源可能会有限制。

流程

您可以以两种方式限制资源使用：

- 通过在项目的默认容器限值中指定资源限值来限制资源使用。
- 在构建配置中通过指定资源限值来限制资源使用。** 在以下示例中，每个 **resources**、**cpu** 和 **memory** 参数都是可选的：

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  resources:
    limits:
      cpu: "100m" 1
      memory: "256Mi" 2
```

1 **cpu** 以 CPU 单元数为单位：**100m** 表示 0.1 个 CPU 单元（100 * 1e-3）。

2 **memory** 以字节为单位：**256Mi** 表示 268435456 字节 (256 * 2 ^ 20)。

不过，如果您的项目定义了配额，则需要以下两项之一：

- 设定了显式 **requests** 的 **resources** 部分：

```
resources:
  requests: 1
    cpu: "100m"
    memory: "256Mi"
```

1 **requests** 对象包含与配额中资源列表对应的资源列表。

- 项目中定义的限值范围，其中 **LimitRange** 对象中的默认值应用到构建过程中创建的 Pod。否则，构建 Pod 创建将失败，说明无法满足配额要求。

2.9.2. 设置最长持续时间

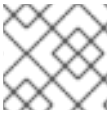
定义 **BuildConfig** 对象时，您可以通过设置 **completionDeadlineSeconds** 字段来定义其最长持续时间。以秒为单位指定，默认情况下不设置。若未设置，则不强制执行最长持续时间。

最长持续时间从构建 Pod 调度到系统中的时间开始计算，并且定义它在多久时间内处于活跃状态，这包括拉取构建器镜像所需的时间。达到指定的超时后，OpenShift Container Platform 将终止构建。

流程

- 要设置最长持续时间，请在 **BuildConfig** 中指定 **completionDeadlineSeconds**。下例显示了 **BuildConfig** 的部分内容，它指定了值为 30 分钟的 **completionDeadlineSeconds** 字段：

```
spec:
  completionDeadlineSeconds: 1800
```



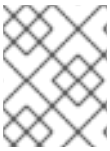
注意

Pipeline 策略选项不支持此设置。

2.9.3. 将构建分配给特定的节点

通过在构建配置的 **nodeSelector** 字段中指定标签，可以将构建定位到在特定节点上运行。**nodeSelector** 值是一组键值对，在调度构建 pod 时与 **Node** 标签匹配。

nodeSelector 值也可以由集群范围的默认值和覆盖值控制。只有构建配置没有为 nodeSelector 定义任何键值对，也没有为 **nodeSelector :{}** 定义显式的空映射值，才会应用默认值。覆盖值将逐个键地替换构建配置中的值。



注意

如果指定的 **NodeSelector** 无法与具有这些标签的节点匹配，则构建仍将无限期地保持在 **Pending** 状态。

流程

- 通过在 **BuildConfig** 的 **nodeSelector** 字段中指定标签，将构建分配到特定的节点上运行，如下例所示：

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  nodeSelector:(1)
    key1: value1
    key2: value2
```

(1) 与此构建配置关联的构建将仅在具有 **key1=value2** 和 **key2=value2** 标签的节点上运行。

2.9.4. 串联构建

对于编译语言（例如 Go、C、C ++ 和 Java），在应用程序镜像中包含编译所需的依赖项可能会增加镜像的大小，或者引入可被利用的漏洞。

为避免这些问题，可以将两个构建串联在一起。一个生成编译工件的构建，另一个构建将工件放置在运行工件的独立镜像中。

在以下示例中，Source-to-Image（S2I）构建与 Docker 构建相结合，以编译工件并将其置于单独的运行时镜像中。

注意

虽然本例串联了 Source-to-Image（S2I） 构建和 Docker 构建，但第一个构建可以使用任何策略来生成包含所需工件的镜像，第二个构建则可以使用任何策略来消耗镜像中的输入内容。

第一个构建获取应用程序源，并生成含有 **WAR** 文件的镜像。镜像推送到 **artifact-image** 镜像流。输出工件的路径取决于使用的 S2I 构建器的 **assemble** 脚本。在这种情况下，它会输出到 **/wildfly/standalone/deployments/ROOT.war**。

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: artifact-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: artifact-image:latest
  source:
    git:
      uri: https://github.com/openshift/openshift-jee-sample.git
      ref: "master"
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: wildfly:10.1
        namespace: openshift
```

第二个构建使用路径指向第一个构建中输出镜像内的 WAR 文件的镜像源。内联 **dockerfile** 将该 **WAR** 文件复制到运行时镜像中。

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: image-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: image-build:latest
  source:
    dockerfile: |-
      FROM jee-runtime:latest
      COPY ROOT.war /deployments/ROOT.war
    images:
    - from: 1
        kind: ImageStreamTag
```

```
      name: artifact-image:latest
    paths: 2
    - sourcePath: /wildfly/standalone/deployments/ROOT.war
      destinationDir: "."
strategy:
  dockerStrategy:
    from: 3
      kind: ImageStreamTag
      name: jee-runtime:latest
triggers:
- imageChange: {}
  type: ImageChange
```

1. **from** 指定 docker 构建应包含来自 **artifact-image** 镜像流的镜像输出，而这是上一个构建的目标。
2. **paths** 指定要在当前 docker 构建中包含目标镜像的哪些路径。
3. 运行时镜像用作 docker 构建的源镜像。

此设置的结果是，第二个构建的输出镜像不需要包含创建 **WAR** 文件所需的任何构建工具。此外，由于第二个构建包含镜像更改触发器，因此每当运行第一个构建并生成含有二进制工件的新镜像时，将自动触发第二个构建，以生成包含该工件的运行时镜像。所以，两个构建表现为一个具有两个阶段的构建。

2.9.5. 修剪构建

默认情况下，生命周期已结束的构建将无限期保留。您可以限制要保留的旧构建数量。

流程

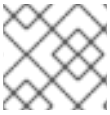
1. 通过为 **BuildConfig** 中的 **successfulBuildsHistoryLimit** 或 **failedBuildsHistoryLimit** 提供正整数值，限制要保留的旧构建的数量，如下例中所示：

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  successfulBuildsHistoryLimit: 2 1
  failedBuildsHistoryLimit: 2 2
```

1. **successfulBuildsHistoryLimit** 将保留最多两个状态为 **completed** 的构建。
2. **failedBuildsHistoryLimit** 将保留最多两个状态为 **failed**、**cancelled** 或 **error** 的构建。

2. 通过以下操作之一来触发构建修剪：
   - 更新构建配置。
   - 等待构建结束其生命周期。

构建按其创建时间戳排序，首先修剪最旧的构建。



注意

管理员可以使用 oc adm 对象修剪命令来手动修剪构建。

2.9.6. 构建运行策略

构建运行策略描述从构建配置创建的构建应运行的顺序。这可以通过更改 **Build** 规格的 **spec** 部分中的 **runPolicy** 字段的值来完成。

还可以通过以下方法更改现有构建配置的 **runPolicy** 值：

- 如果将 **Parallel** 改为 **Serial** 或 **SerialLatestOnly**，并从此配置触发新构建，这会导致新构建需要等待所有并行构建完成，因为串行构建只能单独运行。
- 如果将 **Serial** 更改为 **SerialLatestOnly** 并触发新构建，这会导致取消队列中的所有现有构建，但当前正在运行的构建和最近创建的构建除外。最新的构建接下来运行。

2.10. 在构建中使用红帽订阅

按照以下小节中的内容在 OpenShift Container Platform 上运行授权构建。

2.10.1. 为红帽通用基础镜像创建镜像流标签

要在构建中使用红帽订阅，您可以创建一个镜像流标签来引用通用基础镜像（UBI）。

要让 UBI 在集群中的**每个项目**中都可用，您需要将镜像流标签添加到 **openshift** 命名空间中。否则，若要使其**在一个特定项目中**可用，您要将镜像流标签添加到该项目。

以这种方式使用镜像流标签的好处是，根据安装 pull secret 中的 **registry.redhat.io** 凭证授予对 UBI 的访问权限，而不会向其他用户公开 pull secret。这比要求每个开发人员使用项目中的 **registry.redhat.io** 凭证安装 pull secret 更为方便。

流程

- 要在 **openshift** 命名空间中创建 **ImageStreamTag**，因此所有项目中的开发人员可使用它，请输入：

```
$ oc tag --source=docker registry.redhat.io/ubi8/ubi:latest ubi:latest -n openshift
```

提示

您还可以应用以下 YAML 在 **openshift** 命名空间中创建 **ImageStreamTag** ：

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: ubi
  namespace: openshift
spec:
  tags:
  - from:
      kind: DockerImage
      name: registry.redhat.io/ubi8/ubi:latest
    name: latest
    referencePolicy:
      type: Source
```

- 要在单个项目中创建 **ImageStreamTag**，请输入：

```
$ oc tag --source=docker registry.redhat.io/ubi8/ubi:latest ubi:latest
```

提示

您还可以应用以下 YAML 在单个项目中创建 **ImageStreamTag** ：

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: ubi
spec:
  tags:
  - from:
      kind: DockerImage
      name: registry.redhat.io/ubi8/ubi:latest
    name: latest
    referencePolicy:
      type: Source
```

2.10.2. 将订阅权利添加为构建 secret

使用红帽订阅安装内容的构建需要包括做为一个构件 secret 的权利密钥。

先决条件

您必须可通过您的订阅访问红帽权利。Insights Operator 会自动创建授权 secret。

提示

使用 Red Hat Enterprise Linux(RHEL)7 执行 Entitlement Build 时，在运行任何 **yum** 命令前，必须在 Dockerfile 中包含以下指令：

```
RUN rm /etc/rhsm-host
```

流程

1. 在构建配置的 Docker 策略中将 etc-pki-entitlement secret 添加为构建卷：

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
      name: ubi:latest
    volumes:
    - name: etc-pki-entitlement
      mounts:
      - destinationPath: /etc/pki/entitlement
      source:
        type: Secret
        secret:
          secretName: etc-pki-entitlement
```

2.10.3. 使用 Subscription Manager 运行构建

2.10.3.1. 使用 Subscription Manager 执行 Docker 构建

Docker 策略构建可以使用 Subscription Manager 来安装订阅内容。

先决条件

必须将授权密钥添加为构建策略卷。

流程

使用以下示例 Dockerfile 来通过 Subscription Manager 安装内容：

```
FROM registry.redhat.io/ubi8/ubi:latest
RUN dnf search kernel-devel --showduplicates && \
        dnf install -y kernel-devel
```

2.10.4. 使用 Red Hat Satellite 订阅运行构建

2.10.4.1. 将 Red Hat Satellite 配置添加到构建中

使用 Red Hat Satellite 安装内容的构建必须提供适当的配置，以便从 Satellite 存储库获取内容。

先决条件

- 您必须提供或创建与 **yum** 兼容的存储库配置文件，该文件将从 Satellite 实例下载内容。

  仓库配置示例

```
[test-<name>]
name=test-<number>
baseurl = https://satellite.../content/dist/rhel/server/7/7Server/x86_64/os
enabled=1
gpgcheck=0
```

```
sslverify=0
sslclientkey = /etc/pki/entitlement/...-key.pem
sslclientcert = /etc/pki/entitlement/....pem
```

流程

1. 创建包含 Satellite 存储库配置文件的 **ConfigMap**：

```
$ oc create configmap yum-repos-d --from-file /path/to/satellite.repo
```

2. 将 Satellite 存储库配置和授权密钥添加为构建卷：

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
      name: ubi:latest
    volumes:
    - name: yum-repos-d
      mounts:
      - destinationPath: /etc/yum.repos.d
      source:
        type: ConfigMap
        configMap:
          name: yum-repos-d
    - name: etc-pki-entitlement
      mounts:
      - destinationPath: /etc/pki/entitlement
      source:
        type: Secret
        secret:
          secretName: etc-pki-entitlement
```

2.10.4.2. 使用 Red Hat Satellite 订阅构建 Docker

Docker 策略构建可以使用 Red Hat Satellite 软件仓库来安装订阅内容。

先决条件

- 您已将授权密钥和 Satellite 存储库配置添加为构建卷。

流程

使用以下示例 Dockerfile 来通过 Satellite 安装内容：

```
FROM registry.redhat.io/ubi8/ubi:latest
RUN dnf search kernel-devel --showduplicates && \
        dnf install -y kernel-devel
```

其他资源

- 如何使用 Red Hat Satellite 订阅和使用哪个证书进行构建

2.10.5. 使用 SharedSecret 对象运行授权构建

您可以在一个可以安全地使用来自另外一个命名空间中的一个**Secret** 中的 RHEL 权利的命名空间中，配置和执行构建。

您仍可通过在与 **Build** 对象相同的命名空间中，创建包含订阅凭证的 **Secret** 对象来从 OpenShift 构建中访问 RHEL 权利。但是，在 OpenShift Container Platform 4.10 及之后的版本中，您可以从 OpenShift Container Platform 系统命名空间中的一个 **Secret** 对象来访问您的凭证和证书。您可以使用引用 **Secret** 对象的 **SharedSecret** 自定义资源(CR)实例的 CSI 卷挂载运行授权构建。

此流程依赖于新引入的 Shared Resources CSI Driver 功能，该功能可用于在 OpenShift Container Platform 构建中声明 CSI 卷挂载。它还依赖于 OpenShift Container Platform Insights Operator。

重要

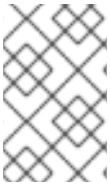
共享资源 CSI 驱动程序和构建 CSI 卷都是技术预览功能，在红帽产品服务等级协议(SLA)中不支持，且可能无法正常工作。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Shared Resources CSI Driver 和 Build CSI Volumes 功能还属于 **TechPreviewNoUpgrade** 功能集，它是当前技术预览功能的子集。您可以在测试集群中启用 **TechPreviewNoUpgrade** 功能集，您可以在生产环境集群中禁用这些功能时完全测试它们。启用此功能集无法撤消并阻止次版本更新。不建议在生产环境集群中使用此功能集。请参阅以下 "Additional resources" 部分，请参阅使用功能门"启用技术预览功能"。

先决条件

- 已使用功能门启用 **TechPreviewNoUpgrade** 功能集。
- 您有一个 **SharedSecret** 自定义资源(CR)实例，它引用 Insights Operator 存储订阅凭证的 **Secret** 对象。
- 您必须具有执行以下操作的权限：
  - 创建构建配置和启动构建。
  - 输入 **oc get sharedsecrets** 命令并返回非空列表来发现哪些 **SharedSecret** CR 实例可用。
  - 确定命名空间中的 **builder** 服务帐户是否可以使用给定的 **SharedSecret** CR 实例。换句话说，您可以运行 **oc adm policy who-can use <identifier of specific SharedSecret>** 来查看是否列出命名空间中的 **builder** 服务帐户。

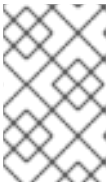


注意

如果没有满足此列表中的最后两个先决条件，则建立或询问某人建立所需的基于角色的访问控制(RBAC)，以便您可以发现 **SharedSecret** CR 实例，并启用服务帐户使用 **SharedSecret** CR 实例。

流程

1. 使用带有 YAML 内容的 **oc apply** 命令，为 **builder** 服务帐户 RBAC 权限授予使用 **SharedSecret** CR 实例：



注意

目前，**kubectl** 和 **oc** 具有硬编码的特殊大小写逻辑，以便将 **use** 动词的使用限制为只限于 pod 安全性相关的角色。因此，您无法使用 **oc create role …** 创建使用 **SharedSecret** CR 实例所需的角色。

使用带有 YAML 角色对象定义的 oc apply -f 命令示例

```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-my-share
  namespace: my-namespace
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedsecrets
    resourceNames:
      - my-share
    verbs:
      - use
EOF
```

2. 使用 **oc** 命令创建与角色关联的 **RoleBinding** ：

oc create rolebinding 命令示例

```
$ oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. 创建可访问 RHEL 权利的 **BuildConfig** 对象。

YAML BuildConfig 对象定义示例

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: my-csi-bc
  namespace: my-csi-app-namespace
spec:
  runPolicy: Serial
  source:
    dockerfile: |
      FROM registry.redhat.io/ubi8/ubi:latest
      RUN ls -la /etc/pki/entitlement
      RUN rm /etc/rhsm-host
      RUN yum repolist --disablerepo=*
      RUN subscription-manager repos --enable rhocp-4.9-for-rhel-8-x86_64-rpms
      RUN yum -y update
      RUN yum install -y openshift-clients.x86_64
  strategy:
    type: Docker
```

```
    dockerStrategy:
      volumes:
        - mounts:
            - destinationPath: "/etc/pki/entitlement"
          name: my-csi-shared-secret
          source:
            csi:
              driver: csi.sharedresource.openshift.io
              readOnly: true
              volumeAttributes:
                sharedSecret: my-share-bc
            type: CSI
```

4. 从 **BuildConfig** 对象启动一个构建，并使用 **oc** 命令跟踪日志。

oc start-build 命令示例

```
$ oc start-build my-csi-bc -F
```

例 2.1. oc start-build 命令的输出示例

注意

以下输出的一些部分已被 **...**替换

```
build.build.openshift.io/my-csi-bc-1 started
Caching blobs under "/var/cache/blobs".

Pulling image registry.redhat.io/ubi8/ubi:latest ...
Trying to pull registry.redhat.io/ubi8/ubi:latest...
Getting image source signatures
Copying blob
sha256:5dcbdc60ea6b60326f98e2b49d6ebcb7771df4b70c6297ddf2d7dede6692df6e
Copying blob
sha256:8671113e1c57d3106acaef2383f9bbfe1c45a26eacb03ec82786a494e15956c3
Copying config
sha256:b81e86a2cb9a001916dc4697d7ed4777a60f757f0b8dcc2c4d8df42f2f7edb3a
Writing manifest to image destination
Storing signatures
Adding transient rw bind mount for /run/secrets/rhsm
STEP 1/9: FROM registry.redhat.io/ubi8/ubi:latest
STEP 2/9: RUN ls -la /etc/pki/entitlement
total 360
drwxrwxrwt. 2 root root  80 Feb  3 20:28 .
drwxr-xr-x. 10 root root 154 Jan 27 15:53 ..
-rw-r--r--. 1 root root   3243 Feb  3 20:28 entitlement-key.pem
-rw-r--r--. 1 root root 362540 Feb  3 20:28 entitlement.pem
time="2022-02-03T20:28:32Z" level=warning msg="Adding metacopy option, configured
globally"
--> 1ef7c6d8c1a
STEP 3/9: RUN rm /etc/rhsm-host
time="2022-02-03T20:28:33Z" level=warning msg="Adding metacopy option, configured
globally"
--> b1c61f88b39
```

```
STEP 4/9: RUN yum repolist --disablerepo=*
Updating Subscription Management repositories.


...

--> b067f1d63eb
STEP 5/9: RUN subscription-manager repos --enable rhocp-4.9-for-rhel-8-x86_64-rpms
Repository 'rhocp-4.9-for-rhel-8-x86_64-rpms' is enabled for this system.
time="2022-02-03T20:28:40Z" level=warning msg="Adding metacopy option, configured
globally"
--> 03927607ebd
STEP 6/9: RUN yum -y update
Updating Subscription Management repositories.

...

Upgraded:
  systemd-239-51.el8_5.3.x86_64       systemd-libs-239-51.el8_5.3.x86_64
  systemd-pam-239-51.el8_5.3.x86_64
Installed:
  diffutils-3.6-6.el8.x86_64            libxkbcommon-0.9.1-1.el8.x86_64
  xkeyboard-config-2.28-1.el8.noarch

Complete!
time="2022-02-03T20:29:05Z" level=warning msg="Adding metacopy option, configured
globally"
--> db57e92ff63
STEP 7/9: RUN yum install -y openshift-clients.x86_64
Updating Subscription Management repositories.

...

Installed:
  bash-completion-1:2.7-5.el8.noarch
  libpkgconf-1.4.2-1.el8.x86_64
  openshift-clients-4.9.0-202201211735.p0.g3f16530.assembly.stream.el8.x86_64
  pkgconf-1.4.2-1.el8.x86_64
  pkgconf-m4-1.4.2-1.el8.noarch
  pkgconf-pkg-config-1.4.2-1.el8.x86_64

Complete!
time="2022-02-03T20:29:19Z" level=warning msg="Adding metacopy option, configured
globally"
--> 609507b059e
STEP 8/9: ENV "OPENSHIFT_BUILD_NAME"="my-csi-bc-1"
"OPENSHIFT_BUILD_NAMESPACE"="my-csi-app-namespace"
--> cab2da3efc4
STEP 9/9: LABEL "io.openshift.build.name"="my-csi-bc-1"
"io.openshift.build.namespace"="my-csi-app-namespace"
COMMIT temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-1:edfe12ca
--> 821b582320b
Successfully tagged temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-
1:edfe12ca
821b582320b41f1d7bab4001395133f86fa9cc99cc0b2b64c5a53f2b6750db91
Build complete, no image push requested
```

```
```

2.10.6. 其他资源

- 使用 Insights Operator 导入简单的内容访问证书
- 使用功能门启用功能
- 管理镜像流
- 构建策略

2.11. 通过策略保护构建

OpenShift Container Platform 中的构建在特权容器中运行。根据所用的构建策略，如果您有权限，可以运行构建来升级其在集群和主机节点上的权限。为安全起见，请限制可以运行构建的人员以及用于这些构建的策略。Custom 构建本质上不如 Source 构建安全，因为它们可以在特权容器内执行任何代码，这在默认情况下是禁用的。请谨慎授予 docker 构建权限，因为 Dockerfile 处理逻辑中的漏洞可能会导致在主机节点上授予特权。

默认情况下，所有能够创建构建的用户都被授予相应的权限，可以使用 docker 和 Source-to-Image (S2I) 构建策略。具有集群管理员特权的用户可启用自定义构建策略,如在全局范围内限制用户使用构建策略部分中所述。

您可以使用授权策略来控制谁能够构建以及他们可以使用哪些构建策略。每个构建策略都有一个对应的构建子资源。用户必须有权创建构建，并在构建策略子资源上创建构建的权限，才能使用该策略创建构建。提供的默认角色用于授予构建策略子资源的 create 权限。

表 2.3. 构建策略子资源和角色

策略	子资源	角色
Docker	builds/docker	system:build-strategy-docker
Source-to-Image	builds/source	system:build-strategy-source
Custom	builds/custom	system:build-strategy-custom
JenkinsPipeline	builds/jenkinspipeline	system:build-strategy-jenkinspipeline

2.11.1. 在全局范围内禁用构建策略访问

要在全局范围内阻止对特定构建策略的访问，请以具有集群管理源权限的用户身份登录，从 **system:authenticated** 组中移除对应的角色，再应用注解 **rbac.authorization.kubernetes.io/autoupdate: "false"** 以防止它们在 API 重启后更改。以下示例演示了如何禁用 Docker 构建策略。

流程

1. 应用 **rbac.authorization.kubernetes.io/autoupdate** 注解：

```
$ oc edit clusterrolebinding system:build-strategy-docker-binding
```

-

输出示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false" (1)
  creationTimestamp: 2018-08-10T01:24:14Z
  name: system:build-strategy-docker-binding
  resourceVersion: "225"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/system%3Abuild-strategy-
docker-binding
  uid: 17b1f3d4-9c3c-11e8-be62-0800277d20bf
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:build-strategy-docker
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
```

(1) 将 **rbac.authorization.kubernetes.io/autoupdate** 注解的值更改为 **"false"**。

2. 移除角色：

```
$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker
system:authenticated
```

3. 确保也从这些角色中移除构建策略子资源：

```
$ oc edit clusterrole admin
```

```
$ oc edit clusterrole edit
```

4. 对于每个角色，指定与要禁用的策略资源对应的子资源。

   a. 为 **admin** 禁用 docker Build 策略：

```
kind: ClusterRole
metadata:
  name: admin
...
- apiGroups:
  - ""
  - build.openshift.io
  resources:
  - buildconfigs
  - buildconfigs/webhooks
  - builds/custom (1)
  - builds/source
  verbs:
```

```
    - create
    - delete
    - deletecollection
    - get
    - list
    - patch
    - update
    - watch
...
```

1 添加 **builds/custom** 和 **builds/source**，以在全局范围内为具有 **admin** 角色的用户禁用 docker 构建。

2.11.2. 在全局范围内限制用户使用构建策略

您可以允许某一组用户使用特定策略来创建构建。

先决条件

- 禁用构建策略的全局访问。

流程

- 将与构建策略对应的角色分配给特定用户。例如，将 **system:build-strategy-docker** 集群角色添加到用户 **devuser**：

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```

警告

如果在集群级别授予用户对 **builds/docker** 子资源的访问权限，那么该用户将能够在他们可以创建构建的任何项目中使用 docker 策略来创建构建。

2.11.3. 在项目范围内限制用户使用构建策略

与在全局范围内向用户授予构建策略角色类似，您只能允许项目中的某一组特定用户使用特定策略来创建构建。

先决条件

- 禁用构建策略的全局访问。

流程

- 将与构建策略对应的角色分配给项目中的特定用户。例如，将 **devproject** 项目中的 **system:build-strategy-docker** 角色添加到用户 **devuser**：

```
$ oc adm policy add-role-to-user system:build-strategy-docker devuser -n devproject
```

2.12. 构建配置资源

使用以下步骤来配置构建设置。

2.12.1. 构建控制器配置参数

build.config.openshift.io/cluster 资源提供以下配置参数。

参数	描述
Build	包含有关如何处理构建的集群范围内信息。规范且唯一有效的名称是 **cluster**。 **spec**：包含构建控制器配置的用户可设置值。
buildDefaults	控制构建的默认信息。 **defaultProxy**：包含所有构建操作的默认代理设置，包括镜像拉取或推送以及源代码下载。 您可以通过设置 **BuildConfig** 策略中的 **HTTP_PROXY**、**HTTPS_PROXY** 和 **NO_PROXY** 环境变量来覆盖值。 **gitProxy**：仅包含 Git 操作的代理设置。如果设置，这将覆盖所有 Git 命令的任何代理设置，例如 **git clone**。 此处未设置的值将从 DefaultProxy 继承。 **env**：一组应用到构建的默认环境变量，条件是构建中不存在指定的变量。 **imageLabels**：应用到生成的镜像的标签列表。您可以通过在 **BuildConfig** 中提供具有相同名称的标签来覆盖默认标签。 **resources**：定义执行构建的资源要求。
ImageLabel	**name**：定义标签的名称。它必须具有非零长度。
buildOverrides	控制构建的覆盖设置。 **imageLabels**：应用到生成的镜像的标签列表。如果您在 **BuildConfig** 中提供了与此表中名称相同的标签，您的标签将会被覆盖。 **nodeSelector**：一个选择器，必须为 true 才能使构建 Pod 适合节点。 **tolerations**：一个容忍度列表，覆盖构建 Pod 上设置的现有容忍度。
BuildList	**items**：标准对象的元数据。

2.12.2. 配置构建设置

您可以通过编辑 **build.config.openshift.io/cluster** 资源来配置构建设置。

流程

- 编辑 **build.config.openshift.io/cluster** 资源：

```
$ oc edit build.config.openshift.io/cluster
```

以下是 **build.config.openshift.io/cluster** 资源的示例：

```
apiVersion: config.openshift.io/v1
kind: Build (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 2
  name: cluster
  resourceVersion: "107233"
  selfLink: /apis/config.openshift.io/v1/builds/cluster
  uid: e2e9cc14-78a9-11e9-b92b-06d6c7da38dc
spec:
  buildDefaults: (2)
    defaultProxy: (3)
      httpProxy: http://proxy.com
      httpsProxy: https://proxy.com
      noProxy: internal.com
    env: (4)
    - name: envkey
      value: envvalue
    gitProxy: (5)
      httpProxy: http://gitproxy.com
      httpsProxy: https://gitproxy.com
      noProxy: internalgit.com
    imageLabels: (6)
    - name: labelkey
      value: labelvalue
    resources: (7)
      limits:
        cpu: 100m
        memory: 50Mi
      requests:
        cpu: 10m
        memory: 10Mi
  buildOverrides: (8)
    imageLabels: (9)
    - name: labelkey
      value: labelvalue
    nodeSelector: (10)
      selectorkey: selectorvalue
    tolerations: (11)
    - effect: NoSchedule
      key: node-role.kubernetes.io/builds
operator: Exists
```

(1) **Build**：包含有关如何处理构建的集群范围内信息。规范且唯一有效的名称是 **cluster**。

(2) **buildDefaults**：控制构建的默认信息。

3. **defaultProxy**：包含所有构建操作的默认代理设置，包括镜像拉取或推送以及源代码下载。
4. **env**：一组应用到构建的默认环境变量，条件是构建中不存在指定的变量。
5. **gitProxy**：仅包含 Git 操作的代理设置。如果设置，这将覆盖所有 Git 命令的任何代理设置，例如 **git clone**。
6. **imageLabels**：应用到生成的镜像的标签列表。您可以通过在 **BuildConfig** 中提供具有相同名称的标签来覆盖默认标签。
7. **resources**：定义执行构建的资源要求。
8. **buildOverrides**：控制构建的覆盖设置。
9. **imageLabels**：应用到生成的镜像的标签列表。如果您在 **BuildConfig** 中提供了与此表中名称相同的标签，您的标签将会被覆盖。
10. **nodeSelector**：一个选择器，必须为 true 才能使构建 Pod 适合节点。
11. **tolerations**：一个容忍度列表，覆盖构建 Pod 上设置的现有容忍度。

2.13. 构建故障排除

使用以下内容来排除构建问题。

2.13.1. 解决资源访问遭到拒绝的问题

如果您的资源访问请求遭到拒绝：

问题

构建失败并显示以下信息：

```
requested access to the resource is denied
```

解决方案

您已超过项目中设置的某一镜像配额。检查当前的配额，并验证应用的限值和正在使用的存储：

```
$ oc describe quota
```

2.13.2. 服务证书生成失败

如果您的资源访问请求遭到拒绝：

问题

如果服务证书生成失败并显示以下信息（服务的 **service.beta.openshift.io/serving-cert-generation-error** 注解包含）：

输出示例

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

解决方案

生成证书的服务不再存在，或者具有不同的 **serviceUID** 。您必须删除旧 secret 并清除服务上的以下注解 **service.beta.openshift.io/serving-cert-generation-error** 和 **service.beta.openshift.io/serving-cert-generation-error-num** 以强制重新生成证书：

```
$ oc delete secret <secret_name>
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```

注意

在用于移除注解的命令中，要移除的注解后面有一个 **-**。

2.14. 为构建设置其他可信证书颁发机构

在从镜像 registry 中拉取镜像时，参照以下部分设置构建可信任的额外证书颁发机构 (CA) 。

此流程要求集群管理员创建 **ConfigMap**，并在 **ConfigMap** 中添加额外的 CA 作为密钥。

- **ConfigMap** 必须在 **openshift-config** 命名空间中创建。
- **domain** 是 **ConfigMap** 中的键，**value** 是 PEM 编码的证书。
  - 每个 CA 必须与某个域关联。域格式是 **hostname[..port]**。
- **ConfigMap** 名称必须在 **image.config.openshift.io/cluster** 集群范围配置资源的 **spec.additionalTrustedCA** 字段中设置。

2.14.1. 在集群中添加证书颁发机构

您可以按照以下流程将证书颁发机构 (CA) 添加到集群，以便在推送和拉取镜像时使用。

先决条件

- 您必须具有集群管理员特权。
- 您必须有权访问 registry 的公共证书，通常是位于 **/etc/docker/certs.d/** 目录中的 **hostname/ca.crt** 文件。

流程

1. 在 **openshift-config** 命名空间中创建一个 **ConfigMap**，其中包含使用自签名证书的 registry 的可信证书。对于每个 CA 文件，确保 **ConfigMap** 中的键是 **hostname[..port]** 格式的容器镜像仓库的主机名：

```
$ oc create configmap registry-cas -n openshift-config \
--from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
--from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. 更新集群镜像配置：

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-cas"}}}' --type=merge
```

2.14.2. 其他资源

- 创建一个 **ConfigMap**
- Secrets 和 **ConfigMaps**
- 配置自定义 PKI

第 3 章 从 JENKINS 迁移到 TEKTON

3.1. 从 JENKINS 迁移到 TEKTON

Jenkins 和 Tekton 广泛用于自动化构建、测试和部署应用和项目的过程。但是，Tekton 是一个云原生 CI/CD 解决方案，它可与 Kubernetes 和 OpenShift Container Platform 无缝配合工作。本文档可帮助您将 Jenkins CI/CD 工作流迁移到 Tekton。

3.1.1. Jenkins 和 Tekton 概念的比较

本节总结了 Jenkins 和 Tekton 中使用的基本术语，并比较了相同的术语。

3.1.1.1. Jenkins 术语

Jenkins 提供声明式和脚本化管道，它们可以使用共享库和插件扩展。Jenkins 中的一些基本术语如下：

- **管道（Pipeline）**：利用 Groovy 语法自动化构建、测试和部署应用的整个流程。
- **节点（Node）**：能够编配或执行脚本化管道的计算机。
- **阶段（Stage）**：在管道中执行的概念上不同的任务子集。插件或用户界面通常使用此块来显示任务的状态或进度。
- **步骤（Step）**：一项任务指定要执行的确切操作，可使用命令或脚本。

3.1.1.2. Tekton 术语

Tekton 使用 YAML 语法用于声明管道，由任务组成。Tekton 中的一些基本术语如下：

- **频道（Pipeline）**：一组串行或并行（或两者）任务。
- **任务（Task）**：作为命令、二进制文件或脚本的步骤序列。
- **管道运行（PipelineRun）**：执行包含一个或多个任务的管道。
- **任务运行（TaskRun）**：通过一个或多个步骤执行任务。

注意

您可以使用一组输入（如参数和工作区）启动 PipelineRun 或 TaskRun，执行会产生一组输出和工件。

- **工作区（Workspace）**：在 Tekton 中，工作区是概念块，用于以下目的：
  - 存储输入、输出和构建工件.
  - 在任务间共享数据的通用空间.
  - 在 secret 中保存的凭证挂载点、配置映射中保存的配置以及机构共享的通用工具。

注意

在 Jenkins 中，没有 Tekton 工作区直接等效的工作区。您可以将控制节点视为工作区，因为它存储克隆的代码存储库、构建历史记录和工件。当作业分配到其他节点时，克隆的代码和生成的工件会存储在该节点中，但构建历史记录由控制节点维护。

3.1.1.3. 概念映射

Jenkins 和 Tekton 的构建块并不匹配，对它们的比较在技术上并不是准确的映射。Jenkins 和 Tekton 中的以下术语和概念一般相对应：

表 3.1. Jenkins 和 Tekton - 基本比较

Jenkins	Tekton
Pipeline	Pipeline 和 PipelineRun
Stage	任务
Step	一个任务中的一个步骤

3.1.2. 将示例管道从 Jenkins 迁移到 Tekton

本节提供了 Jenkins 和 Tekton 中的等效管道示例，可帮助您将构建、测试和部署管道从 Jenkins 迁移到 Tekton。

3.1.2.1. Jenkins 管道

考虑在 Groovy 中编写的 Jenkins 管道，用于构建、测试和部署：

```
pipeline {
  agent any
  stages {
      stage('Build') {
          steps {
              sh 'make'
          }
      }
      stage('Test'){
          steps {
              sh 'make check'
              junit 'reports/**/*.xml'
          }
      }
      stage('Deploy') {
          steps {
              sh 'make publish'
          }
      }
  }
}
```

3.1.2.2. Tekton pipeline

在 Tekton 中，Jenkins 管道的等效示例由三个任务组成，每个任务都可以使用 YAML 语法来声明编写：

build 任务示例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-build
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make"]
    workingDir: $(workspaces.source.path)
```

test 任务示例：

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make check"]
    workingDir: $(workspaces.source.path)
  - image: junit-report-image
    script: |
      #!/usr/bin/env bash
      junit-report reports/**/*.xml
    workingDir: $(workspaces.source.path)
```

deploy 任务示例：

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
  - name: source
  steps:
  - image: my-deploy-image
    command: ["make deploy"]
    workingDir: $(workspaces.source.path)
```

您可以按顺序组合三个任务组成 Tekton 管道：

示例：用于构建、测试和部署的 Tekton 管道

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: myproject-pipeline
spec:
  workspaces:
  - name: shared-dir
  tasks:
  - name: build
    taskRef:
      name: myproject-build
    workspaces:
    - name: source
      workspace: shared-dir
  - name: test
    taskRef:
      name: myproject-test
    workspaces:
    - name: source
      workspace: shared-dir
  - name: deploy
    taskRef:
      name: myproject-deploy
    workspaces:
    - name: source
      workspace: shared-dir
```

3.1.3. 从 Jenkins 插件迁移到 Tekton Hub 任务

您可以使用 插件 来扩展 Jenkins 的功能。要在 Tekton 中实现类似的可扩展性，请使用 Tekton Hub 中的任何可用任务。

例如，请考虑 Tekton Hub 中可用的 git-clone 任务，它对应于 Jenkins 的 git 插件。

示例：Tekton Hub 中的 git-clone 任务

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
 name: demo-pipeline
spec:
 params:
   - name: repo_url
   - name: revision
 workspaces:
   - name: source
 tasks:
   - name: fetch-from-git
     taskRef:
       name: git-clone
     params:
       - name: url
         value: $(params.repo_url)
       - name: revision
         value: $(params.revision)
```

```
    workspaces:
    - name: output
      workspace: source
```

3.1.4. 使用自定义任务和脚本扩展 Tekton 功能

在 Tekton 中，如果您在 Tekton Hub 中找不到正确的任务，或需要对任务进行更大的控制，您可以创建自定义任务和脚本来扩展 Tekton 的功能。

示例：运行 maven test 命令的自定义任务

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-maven-image
    command: ["mvn test"]
    workingDir: $(workspaces.source.path)
```

示例：通过提供自定义 shell 脚本的路径

```
...
steps:
  image: ubuntu
  script: |
      #!/usr/bin/env bash
      /workspace/my-script.sh
...
```

示例：在 YAML 文件中写入一个自定义 Python 脚本

```
...
steps:
  image: python
  script: |
      #!/usr/bin/env python3
      print(“hello from python!”)
...
```

3.1.5. Jenkins 和 Tekton 执行模型的比较

Jenkins 和 Tekton 提供了类似的功能，但在架构和执行方面有所不同。本节概述了这两种执行模型的简要比较。

表 3.2. Jenkins 和 Tekton 中的执行模型比较

Jenkins	Tekton
Jenkins 有一个控制节点。Jenkins 集中执行管道和步骤，或者编排其他节点上运行的作业。	Tekton 是无服务器且分布式的，它的执行并不依赖于一个中心系统。
容器由控制节点通过管道启动。	Tekton 采用"容器先行"方法，其中每一步骤都作为容器集中运行的容器执行（等同于 Jenkins 中的节点）。
使用插件可实现可扩展性。	使用 Tekton Hub 中的任务或创建自定义任务和脚本来实现可扩展性。

3.1.6. 常见使用案例示例

Jenkins 和 Tekton 都提供通用 CI/CD 用例的功能，例如：

- 使用 maven 编译、构建和部署镜像
- 使用插件扩展核心功能
- 重新使用可共享库和自定义脚本

3.1.6.1. 在 Jenkins 和 Tekton 中运行 maven 管道

您可以在 Jenkins 和 Tekton 工作流中使用 maven 来编译、构建和部署镜像。要将现有 Jenkins 工作流映射到 Tekton，请考虑以下示例：

示例：编译并构建镜像，并使用 Jenkins 中的 maven 将它部署到 OpenShift

```
#!/usr/bin/groovy
node('maven') {
   stage 'Checkout'
   checkout scm

   stage 'Build'
   sh 'cd helloworld && mvn clean'
   sh 'cd helloworld && mvn compile'

   stage 'Run Unit Tests'
   sh 'cd helloworld && mvn test'

   stage 'Package'
   sh 'cd helloworld && mvn package'

   stage 'Archive artifact'
   sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
   archive 'helloworld/target/*.war'

   stage 'Create Image'
   sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
   sh 'oc new-project helloworldproject'
```

```
    sh 'oc project helloworldproject'
    sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
    sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

    stage 'Deploy'
    sh 'oc new-app helloworld/jboss-eap70-deploy.json' }
```

示例：编译并构建镜像，并使用 Tekton 中的 maven 将它部署到 OpenShift。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: $(params.revision)
    - name: mvn-build
      taskRef:
        name: maven
      runAfter:
        - fetch-repo
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: maven-settings
          workspace: maven-settings
      params:
        - name: CONTEXT_DIR
          value: "$(params.context-path)"
        - name: GOALS
          value: ["-DskipTests", "clean", "compile"]
    - name: mvn-tests
      taskRef:
```

```
      name: maven
    runAfter:
      - mvn-build
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["test"]
  - name: mvn-package
    taskRef:
      name: maven
    runAfter:
      - mvn-tests
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["package"]
  - name: create-image-and-deploy
    taskRef:
      name: openshift-client
    runAfter:
      - mvn-package
    workspaces:
      - name: manifest-dir
        workspace: shared-workspace
      - name: kubeconfig-dir
        workspace: kubeconfig-dir
    params:
      - name: SCRIPT
        value: |
          cd "$(params.context-path)"
          mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
          oc new-project helloworldproject
          oc project helloworldproject
          oc process -f jboss-eap70-binary-build.json | oc create -f -
          oc start-build eap-helloworld-app --from-dir=artifacts/
          oc new-app jboss-eap70-deploy.json
```

3.1.6.2. 使用插件扩展 Jenkins 和 Tekton 的核心功能

Jenkins 利用了其广泛的用户群来多年开发的大量插件生态系统。您可以在 Jenkins 插件索引 中搜索和浏览插件。

Tekton 还有许多任务由社区和企业用户开发并贡献。Tekton Hub 中提供了可重复使用 Tekton 任务的公开可用目录。

另外，Tekton 在其核心功能中纳入了 Jenkins 生态系统的许多插件。例如，授权是 Jenkins 和 Tekton 中的关键功能。虽然 Jenkins 使用基于角色的访问控制插件来确保授权，Tekton 使用 OpenShift 的内置基于角色的访问控制系统。

3.1.6.3. 在 Jenkins 和 Tekton 中共享可重复使用的代码

Jenkins 共享库 为 Jenkins 管道的各部分提供可重复使用的代码。该库在 Jenkinsfile 之间共享，以创建高度模块化的管道，而不重复代码。

虽然 Tekton 中没有 Jenkins 共享库直接对应的，但您可以使用 Tekton Hub 中的任务实现类似的工作流，从而结合使用自定义任务和脚本。

3.1.7. 其他资源

- 基于角色的访问控制

第 4 章 PIPELINES

4.1. RED HAT OPENSHIFT PIPELINES 发行注记

Red Hat OpenShift Pipelines 是基于 Tekton 项目的一个云原生 CI/CD 环境，它提供：

- 标准 Kubernetes 原生管道定义 (CRD)。
- 无需 CI 服务器管理开销的无服务器管道。
- 使用任何 Kubernetes 工具（如 S2I、Buildah、JIB 和 Kaniko）构建镜像。
- 不同 Kubernetes 发布系统间的可移植性。
- 用于与管道交互的强大 CLI。
- 使用 OpenShift Container Platform Web 控制台的 **Developer** 视角集成用户体验。

如需了解 Red Hat OpenShift Pipelines 的概述，请参阅了解 OpenShift Pipelines。

4.1.1. 兼容性和支持列表

这个版本中的一些功能当前还只是一个技术预览。它们并不适用于在生产环境中使用。

在下表中，被标记为以下状态的功能：

TP	技术预览
GA	公开发行

表 4.1. 兼容性和支持列表

Red Hat OpenShift Pipelines 版本	组件版本							OpenShift 版本	支持状态
Operator	Pipelines	触发器	CLI	目录	链	Hub	Pipelines 作为代码		
1.10	0.44.x	0.23.x	0.30.x	不适用	0.15.x (TP)	1.12.x (TP)	0.17.x (GA)	4.10, 4.11, 4.12, 4.13	GA

Red Hat OpenShift Pipelines 版本	组件版本							OpenShift 版本	支持状态
1.9	0.41.x	0.22.x	0.28.x	不适用	0.13.x (TP)	1.11.x (TP)	0.15.x (GA)	4.10, 4.11, 4.12, 4.13	GA
1.8	0.37.x	0.20.x	0.24.x	不适用	0.9.0 (TP)	1.8.x (TP)	0.10.x (TP)	4.10, 4.11, 4.12	GA
1.7	0.33.x	0.19.x	0.23.x	0.33	0.8.0 (TP)	1.7.0 (TP)	0.5.x (TP)	4.9, 4.10, 4.11	GA
1.6	0.28.x	0.16.x	0.21.x	0.28	N/A	N/A	N/A	4.9	GA
1.5	0.24.x	0.14.x (TP)	0.19.x	0.24	N/A	N/A	N/A	4.8	GA
1.4	0.22.x	0.12.x (TP)	0.17.x	0.22	N/A	N/A	N/A	4.7	GA

另外，在 ARM 硬件上运行 Red Hat OpenShift Pipelines 当前只是一个技术预览。

如果您有疑问或希望提供反馈信息，请向产品团队发送邮件 pipelines-interest@redhat.com。

4.1.2. 使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

4.1.3. Red Hat OpenShift Pipelines 正式发行（GA）1.10 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.10 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.3.1. 新功能

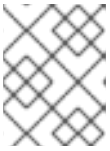
除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.10 中的新内容。

4.1.3.1.1. Pipelines

- 在这个版本中，您可以在 **PipelineRun** 或 **TaskRun** pod 模板中指定环境变量来覆盖或附加任务

或步骤中配置的变量。另外，您可以在默认 pod 模板中指定环境变量，以对所有 **PipelineRuns** 和 **TaskRuns** 全局使用这些变量。在这个版本中，添加了一个名为 **forbidden-envs** 的新默认配置来过滤环境变量，同时从 pod 模板传播。

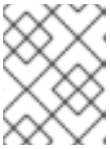
- 在这个版本中，管道中的自定义任务会被默认启用。



注意

要禁用此次更新，请在 **feature-flags** 配置自定义资源中将 **enable-custom-tasks** 标志设置为 **false**。

- 在这个版本中，支持 **v1beta1.CustomRun** API 版本进行自定义任务。
- 在这个版本中，增加了对 **PipelineRun** reconciler 的支持，以创建自定义运行。例如，从 **PipelineRuns** 创建的自定义 **TaskRuns** 现在可以使用 **v1beta1.CustomRun** API 版本而不是 **v1alpha1.Run**，如果 **custom-task-version** 功能标记被设置为 **v1beta1**，而不是默认值 **v1alpha1**。



注意

您需要更新自定义任务控制器，以侦听 ***v1beta1.CustomRun** API 版本，而不是 ***v1alpha1.Run** 以响应 **v1beta1.CustomRun** 请求。

- 在这个版本中，在 **v1beta1.TaskRun** 和 **v1.TaskRun** 规格中添加了一个新的 **retries** 字段。

4.1.3.1.2. 触发器

- 在这个版本中，触发器支持创建 **v1** API 版本的 **Pipelines**, **Tasks**, **PipelineRuns**, 和 **TaskRuns** 对象，以及 **v1beta1** API 版本的 **CustomRun** 对象。
- 在这个版本中，GitHub Interceptor 会阻止执行拉取请求触发器，除非由拥有者调用或所有者使用可配置的注释。

注意

要启用或禁用此更新，请在 GitHub Interceptor 配置文件中将 **githubOwners** 参数的值设置为 **true** 或 **false**。

- 在这个版本中，GitHub Interceptor 能够添加以逗号分隔的、用于推送和拉取请求事件更改的所有文件的列表。更改的文件列表添加到顶层 extensions 字段中事件有效负载的 **changed_files** 属性中。
- 在这个版本中，TLS 的 **MinVersion** 改为 **tls.VersionTLS12**，以便在启用了联邦信息处理标准 (FIPS) 模式时在 OpenShift Container Platform 上运行触发器。

4.1.3.1.3. CLI

- 在这个版本中，增加了对在启动一个 **Task**, **ClusterTask** 或 **Pipeline** 时传递一个 CSI（Container Storage Interface）文件作为工作区的功能。
- 在这个版本中，为与任务、管道运行和任务运行资源关联的所有 CLI 命令添加了 **v1** API 支持。Tekton CLI 适用于这些资源的 **v1beta1** 和 **v1** API。
- 在这个版本中，增加了对 **start** 和 **describe** 命令中的对象类型参数的支持。

4.1.3.1.4. Operator

- 在这个版本中，在可选管道属性中添加了 **default-forbidden-env** 参数。参数包括禁止的环境变量，在通过 pod 模板提供时不应传播它们。
- 在这个版本中，增加了对 Tekton Hub UI 中的自定义徽标的支持。要添加自定义徽标，请在 Tekton Hub CR 中将 **customLogo** 参数的值设置为 base64 编码 URI。
- 在这个版本中，git-clone 任务的版本号增加到 0.9。

4.1.3.1.5. Tekton Chains

重要

Tekton 链只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，在测试时为 **PipelineRun** 和 **TaskRun** 添加注解和标签。
- 此更新添加了一个名为 **slsa/v1** 的新格式，它生成与以 **in-toto** 格式请求时生成的相同度。
- 在这个版本中，Sigstore 功能从实验性功能中移出。
- 在这个版本中，**predicate.materials** 函数包括来自 **TaskRun** 对象的所有步骤和 sidecar 的镜像 URI 和摘要信息。

4.1.3.1.6. Tekton Hub

重要

Tekton Hub 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，支持在集群中安装、升级或降级 **v1** API 版本的 Tekton 资源。
- 在这个版本中，支持添加自定义徽标来代替 UI 中的 Tekton Hub 徽标。
- 在这个版本中，对 **tkn hub install** 命令进行了扩展，增加了一个 **--type artifact** 标记，它会从 Artifact Hub 获取资源并将其安装到集群中。
- 在这个版本中，增加了将 tier, catalog, 和 org 的信息作为标签添加到从 Artifact Hub 安装到您的集群中的资源的支持。

4.1.3.1.7. Pipelines 作为代码（Pipelines as Code）

- 这个版本增强了传入的 Webhook 支持。对于在 OpenShift Container Platform 集群上安装的 GitHub 应用程序，您不需要为传入的 webhook 提供 **git_provider** 规格。相反，Pipelines as Code 会检测 secret，并将其用于传入的 Webhook。

- 在这个版本中，您可以使用同一令牌从带有非默认分支的 GitHub 上的同一主机获取远程任务。
- 在这个版本中，Pipelines 作为代码支持 Tekton **v1** 模板。您可以使用 **v1** 和 **v1beta1** 模板，Pipelines 作为 PR 生成的代码读取。PR 在集群上以 **v1** 的形式创建。
- 在此次更新之前，当 OpenShift 命名空间中找不到运行时模板时，OpenShift 控制台 UI 将使用硬编码的管道运行模板作为回退模板。在这个版本中，**pipelines-as-code** 配置映射中提供了一个新的默认管道运行模板，名为 **pipelines-as-code-template-default** 供控制台使用。
- 在这个版本中，Pipelines as Code 支持 Tekton Pipelines 0.44.0 最小状态。
- 在这个版本中，Pipelines as Code 支持 Tekton **v1** API，这意味着 Pipelines as Code 现在与 Tekton v0.44 及更新的版本兼容。
- 在这个版本中，除了为 OpenShift 和 Tekton 仪表板为 k8s 配置控制台外，您还可以配置自定义控制台仪表板。
- 在这个版本中，Pipelines as Code 会检测使用 **tkn pac create repo** 命令启动的 GitHub 应用程序的安装，如果全局安装，则不需要 GitHub Webhook。
- 在此次更新之前，如果 **PipelineRun** 执行出现错误，而不是附加到 **PipelineRun** 的任务，Pipelines as Code 无法正确报告失败。在这个版本中，Pipelines as Code 会在无法创建 **PipelineRun** 时在 GitHub 检查中正确报告错误。
- 在这个版本中，Pipelines as Code 包含一个 **target_namespace** 变量，它扩展至执行 **PipelineRun** 的当前运行命名空间。
- 在这个版本中，Pipelines 作为代码可让您绕过 CLI bootstrap GitHub 应用程序中的 GitHub Enterprise 问题。
- 在这个版本中，当找不到存储库 CR 时，Pipelines as Code 不会报告错误。
- 在这个版本中，如果找到多个管道使用相同名称运行，Pipelines as Code 会报告一个错误。

4.1.3.2. 可能会造成问题的更改

- 在这个版本中，**tkn** 命令的早期版本与 Red Hat OpenShift Pipelines 1.10 不兼容。
- 在这个版本中，从 Tekton CLI 删除了对 **Cluster** 和 **CloudEvent** 管道资源的支持。您不能使用 **tkn pipelineresource create** 命令创建管道资源。另外，任务、集群任务或管道的 **start** 命令中不再支持管道资源。
- 在这个版本中，从 Tekton 链中删除 **tekton** 作为公认格式。

4.1.3.3. 弃用和删除的功能

- 在 Red Hat OpenShift Pipelines 1.10 中，**ClusterTask** 命令现已弃用，计划在以后的发行版本中删除。此更新中还弃用了 **tkn task create** 命令。
- 在 Red Hat OpenShift Pipelines 1.10 中，**tkn task start** 命令中使用的标记 **-i** 和 **-o** 现已弃用，因为 **v1** API 不支持管道资源。
- 在 Red Hat OpenShift Pipelines 1.10 中，**tkn pipeline start** 命令一起使用的标志 **-r** 已被弃用，因为 **v1** API 不支持管道资源。

- Red Hat OpenShift Pipelines 1.10 更新将 **openshiftDefaultEmbeddedStatus** 参数设置为 **both**，带有 **full** 和 **minimal** 嵌入式状态。更改默认嵌入式状态的标记也已弃用，并将被删除。另外，管道默认嵌入式状态将在以后的版本中更改为 **minimal**。

4.1.3.4. 已知问题

- 这个版本包括以下向后兼容的更改：
  - 删除 **PipelineResources** 集群
  - 删除 **PipelineResources** 云事件
- 如果在集群升级后管道指标功能无法正常工作，请运行以下命令作为临时解决方案：

```
$ oc get tektoninstallersets.operator.tekton.dev | awk '/pipeline-main-static/ {print $1}' | xargs oc delete tektoninstallersets
```

- 在这个版本中，IBM Power、IBM Z 和 {linuxoneProductName} 不支持使用外部数据库，如 Crunchy PostgreSQL。反之，使用默认的 Tekton Hub 数据库。

4.1.3.5. 修复的问题

- 在此次更新之前，**opc pac** 命令会生成一个运行时错误，而不是显示任何帮助。在这个版本中修复了 **opc pac** 命令来显示帮助信息。
- 在此次更新之前，运行 **tkn pac create repo** 命令创建仓库需要 Webhook 详情。在这个版本中，**tkn-pac create repo** 命令在安装 GitHub 应用程序时不会配置 Webhook。
- 在此次更新之前，当 Tekton Pipelines 创建 **PipelineRun** 资源时遇到问题时，Pipelines as Code 不会报告管道运行创建错误。例如，管道运行中的不存在的任务不会显示任何状态。在这个版本中，Pipelines as Code 显示来自 Tekton Pipelines 的正确错误消息以及缺少的任务。
- 在这个版本中，UI 页面在成功身份验证后重定向。现在，您会被重定向到您试图登录到 Tekton Hub 的同一页面。
- 在这个版本中修复了带有标志 **--all-namespaces** 和 **--output=yaml** 的 **list** 命令，用于集群任务、单个任务和管道。
- 在这个版本中，删除了 **repo.spec.url** URL 末尾的正斜杠，使其与来自 GitHub 的 URL 匹配。
- 在此次更新之前，**marshalJSON** 函数不会编译对象列表。在这个版本中，**marshalJSON** 函数会对对象列表进行 marshals 处理。
- 在这个版本中，Pipelines 作为代码可让您绕过 CLI bootstrap GitHub 应用程序中的 GitHub Enterprise 问题。
- 在这个版本中，当您的存储库有超过 100 个用户时，GitHub collaborator 会检查。
- 在这个版本中，任务或管道的 **sign** 和 **verify** 命令现在可以在没有 kubernetes 配置文件的情况下正常工作。
- 在这个版本中，如果在命名空间中跳过修剪器，Tekton Operator 会清理保留的修剪器 cron 作业。
- 在此次更新之前，API **ConfigMap** 对象不会更新为目录刷新间隔配置的值。在这个版本中修复了 Tekon Hub CR 中的 **CATALOG_REFRESH_INTERVAL** API。

- 在这个版本中，在更改 **EmbeddedStatus** 功能标记时，**PipelineStatus** 的协调已被修复。在这个版本中重置以下参数：
  - **status.runs** 和 **status.taskruns** 参数为 **nil**，带有 **minimal EmbeddedStatus**
  - **status.childReferences** 参数为 **nil**，带有 **full EmbeddedStatus**
- 在这个版本中，添加了一个到 **ResolutionRequest** CRD 的转换配置。在这个版本中，可以正确地配置从 **v1alpha1.ResolutionRequest** 请求到 **v1beta1.ResolutionRequest** 请求的转换。
- 在这个版本中会检查与一个管道任务关联的重复工作区。
- 在这个版本中修复了在代码中启用解析器的默认值。
- 在这个版本中，使用解析器修复了 **TaskRef** 和 **PipelineRef** 名称转换的问题。

4.1.3.6. Red Hat OpenShift Pipelines 正式发行（GA）1.10.1 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.10.1 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.3.6.1. 修复了 Pipelines as Code 的问题

- 在此次更新之前，如果来自有效负载的源分支信息包含 **refs/heads/**，但用户配置的目标分支仅包含分支名称，**main** 在 CEL 表达式中，推送请求将失败。在这个版本中，Pipelines as Code 会传递推送请求，并在有效负载中基本分支或目标分支具有 **refs/heads/** 时触发管道。
- 在此次更新之前，当无法创建 **PipelineRun** 对象时，从 Tekton 控制器接收的错误不会被报告给用户。在这个版本中，Pipelines as Code 会向 GitHub 界面报告错误消息，以便用户可以对错误进行故障排除。Pipelines as Code 还会报告管道执行过程中发生的错误。
- 在这个版本中，当因为基础架构问题而无法在 OpenShift Container Platform 集群中创建 secret 时，Pipelines as Code 不会将 secret 回滚到 GitHub 检查接口。
- 在这个版本中，删除了从 Red Hat OpenShift Pipelines 不再使用的已弃用的 API。

4.1.3.7. Red Hat OpenShift Pipelines 正式发布 1.10.2 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.10.2 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.3.7.1. 修复的问题

在此次更新之前，Tekton Operator 中的一个问题会阻止用户将 **enable-api-fields** 标志的值设置为 **beta**。在这个版本中解决了这个问题。现在，您可以在 **TektonConfig** CR 中将 **enable-api-fields** 标志的值设置为 **beta**。

4.1.3.8. Red Hat OpenShift Pipelines 正式发布 1.10.3 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.10.3 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.3.8.1. 修复的问题

在此次更新之前，Tekton Operator 不会公开任何自定义的性能配置字段。在这个版本中，作为集群管理员，您可以根据您的需要自定义 **TektonConfig** CR 中的以下性能配置字段：

- **disable-ha**
- **bucket**
- **kube-api-qps**
- **kube-api-burst**
- **threads-per-controller**

4.1.3.9. Red Hat OpenShift Pipelines 正式发行 1.10.4 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.10.4 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.3.9.1. 修复的问题

- 在这个版本中解决了管道运行中的 **PipelineRef** 字段的捆绑包解析器转换问题。现在，转换功能会在转换后将 **kind** 字段的值设置为 **Pipeline** 。
- 在此次更新之前，**pipelinerun.timeouts** 字段被重置为 **timeout.pipeline** 值，忽略 **timeout.tasks** 和 **timeout.finally** 值。在这个版本中解决了这个问题，并为 **PipelineRun** 资源设置正确的默认超时值。
- 在此次更新之前，控制器日志包含不必要的数据。在这个版本中解决了这个问题。

4.1.3.10. Red Hat OpenShift Pipelines 正式发布（GA）1.10.5 发行注记

在这个版本中，除了 4.11、4.12 和 4.13 外，Red Hat OpenShift Pipelines 正式发布 (GA) 1.10.5 包括在 OpenShift Container Platform 4.10 中。

重要

Red Hat OpenShift Pipelines 1.10.5 仅适用于 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 上的 **pipelines-1.10** 频道。它不适用于任何 OpenShift Container Platform 版本的 **latest** 频道。

4.1.3.10.1. 修复的问题

- 在此次更新之前，使用 **oc** 和 **tkn** 命令无法列出或删除大型管道运行。在这个版本中，通过压缩导致这个问题的巨页注解解决了这个问题。请注意，如果管道运行在压缩后仍然太大，则还会出现同样的错误。
- 在此次更新之前，只有 **pipelineRun.spec.taskRunSpecs[].podTemplate** 对象中指定的 pod 模板才会被视为管道运行。在这个版本中，**pipelineRun.spec.podTemplate** 对象中指定的 pod 模板也会被考虑，并与 **pipelineRun.spec.taskRunSpecs[].podTemplate** 对象中指定的模板合并。

4.1.4. Red Hat OpenShift Pipelines 正式发布 1.9 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.9 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.4.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.9 中的新内容。

4.1.4.1.1. Pipelines

- 在这个版本中，您可以指定管道参数，并产生数组和对象字典形式。
- 在这个版本中，为您的工作区提供 Container Storage Interface (CSI) 和投射卷支持。
- 在这个版本中，您可以在定义管道步骤时指定 **stdoutConfig** 和 **stderrConfig** 参数。定义这些参数有助于将标准输出和标准错误（与步骤相关联）捕获到本地文件。
- 在这个版本中，您可以在 **steps[].onError** 事件处理程序中添加变量，例如 **$(params.CONTINUE)**。
- 在这个版本中，您可以使用 **PipelineResults** 定义中的 **finally** 任务的输出。例如，**$(finally.<pipelinetask-name>.result.<result-name>)**，其中 **<pipelinetask-name>** 代表管道任务名称，**<result-name>** 代表结果名称。
- 在这个版本中，支持任务运行的任务级资源要求。
- 在这个版本中，您不需要根据名称重新创建在管道和定义的任务之间共享的参数。这个版本是开发人员预览功能的一部分。
- 在这个版本中，增加了对远程解析的支持，如内置 git、集群、捆绑包和 hub 解析器。

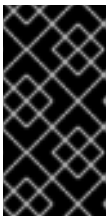
4.1.4.1.2. 触发器

- 在这个版本中，添加了 **Interceptor** CRD 来定义 **NamespacedInterceptor**。您可以在触发器的拦截器引用的 **kind** 部分，或 **EventListener** 规格中使用 **NamespacedInterceptor**。
- 这个版本启用了 **CloudEvents**。
- 在这个版本中，您可以在定义触发器时配置 Webhook 端口号。
- 在这个版本中，支持使用触发器 **eventID** 作为 **TriggerBinding** 的输入。
- 在这个版本中，支持为 **ClusterInterceptor** 服务器验证和轮转证书。
    - 触发器对核心拦截器执行证书验证，并在其证书过期时将新证书轮转到 **ClusterInterceptor**。

4.1.4.1.3. CLI

- 在这个版本中，支持在 **describe** 命令中显示注释。
- 此更新支持在 **pr describe** 命令中显示管道、任务和超时。
- 在这个版本中，添加了在 **pipeline start** 命令中提供管道、任务和超时的标志。
- 在这个版本中，支持在任务和管道的 **describe** 命令中显示工作区（可选或强制）存在。
- 在这个版本中，添加了 **timestamp** 标记来显示带有时间戳的日志。

- 在这个版本中，添加了一个新的标记 **--ignore-running-pipelinerun**，它会忽略删除与 **PipelineRun** 关联的 **TaskRun**。
- 在这个版本中，增加了对实验性命令的支持。在这个版本中，对 **tkn** CLI 工具增加了实验性的子命令 **sign** 和 **verify**。
- 在这个版本中，Zsh (Zsh) 完成功能可在不生成任何文件的情况下使用。
- 此更新引入了一个名为 **opc** 的新 CLI 工具。预计即将推出的发行版本会将 **tkn** CLI 工具替换为 **opc**。



重要

  - 新的 CLI 工具 **opc** 是一个技术预览功能。
  - **opc** 将是一个带有额外 Red Hat OpenShift Pipelines 特定功能的 **tkn** 的替代，这些功能不一定适合 **tkn**。

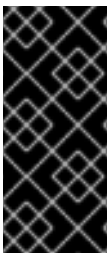
4.1.4.1.4. Operator

- 在这个版本中，Pipelines as Code 会被默认安装。您可以使用 **-p** 标志禁用 Pipelines as Code:

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": false}}}}}'
```

- 在这个版本中，您还可以在 **TektonConfig** CRD 中修改 Pipelines 作为代码配置。
- 在这个版本中，如果禁用了开发人员视角，Operator 不会安装与开发人员控制台相关的自定义资源。
- 在这个版本中，包括对 Bitbucket 服务器和 Bitbucket 云的 **ClusterTriggerBinding** 支持，可帮助您在整个集群中重复使用 **TriggerBinding** 。

4.1.4.1.5. 解析器



重要

解析器只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，您可以在 **TektonConfig** CRD 中配置管道解析器。您可以启用或禁用这些管道解析器： **enable-bundles-resolver**,**enable-cluster-resolver**,**enable-git-resolver**, 和 **enable-hub-resolver**。

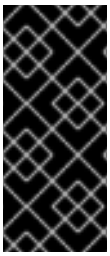
```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    enable-bundles-resolver: true
```

```
    enable-cluster-resolver: true
    enable-git-resolver: true
    enable-hub-resolver: true
...
```

您还可以在 **TektonConfig** 中提供特定于解析器的配置。例如，您可以使用 **map[string]string** 格式定义以下字段来为单个解析器设置配置：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines
    cluster-resolver-config:
      default-namespace: test
    git-resolver-config:
      server-url: localhost.com
    hub-resolver-config:
      default-tekton-hub-catalog: tekton
...
```

4.1.4.1.6. Tekton Chains



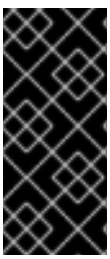
重要

Tekton 链只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在此次更新之前，只有开放容器项目 (OCI) 镜像可以作为 in-toto provenance 代理的 **TaskRun** 输出被支持。在这个版本中，通过使用以下后缀增加了 in-toto provenance 元数据作为输出的支持：**ARTIFACT_URI** 和 **ARTIFACT_DIGEST**。
- 在此次更新之前，只支持 **TaskRun** attestations。在这个版本中，还添加了对 **PipelineRun** attestations 的支持。
- 在这个版本中，添加了对 Tekton Chains 的支持，以获取 pod 模板中的 **imgPullSecret** 参数。在这个版本中，您可以基于每个管道运行或任务运行配置存储库身份验证，而无需修改服务帐户。

4.1.4.1.7. Tekton Hub



重要

Tekton Hub 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，作为管理员，您可以使用外部数据库，如带有 Tekton Hub 的 Crunchy PostgreSQL，而不是使用默认的 Tekton Hub 数据库。这个版本可帮助您执行以下操作：
  - 指定用于 Tekton Hub 的外部数据库的协调
  - 禁用 Operator 部署的默认 Tekton Hub 数据库
- 在这个版本中，**config.yaml** 从外部 Git 存储库删除依赖项，并将完整的配置数据移到 API **ConfigMap** 中。这个版本可帮助管理员执行以下操作：
  - 在 Tekton Hub 自定义资源中添加配置数据，如类别、目录、范围和 defaultScope。
  - 修改集群中的 Tekton Hub 配置数据。所有修改都会在 Operator 升级时保留。
  - 更新 Tekton Hub 目录列表
  - 更改 Tekton Hub 的类别

注意

如果没有添加任何配置数据，您可以使用 Tekton Hub 配置的 API **ConfigMap** 中的默认数据。

4.1.4.1.8. Pipelines 作为代码（Pipelines as Code）

- 在这个版本中，增加了对 **Repository** CRD 中并发限制的支持，以定义一次为仓库运行的最大 **PipelineRuns** 数。来自拉取请求或推送事件的 **PipelineRuns** 以字母顺序排列。
- 在这个版本中，添加了一个新的命令 **tkn pac logs**，用于显示仓库的最新管道运行的日志。
- 在这个版本中，支持在文件路径上实现高级事件匹配，以便推送和拉取请求到 GitHub 和 GitLab。例如，只有当一个路径在 **docs** 目录中的任何 markdown 文件中都更改时，您才可以使用 Common Expression Language (CEL) 来运行管道。

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
   event == "pull_request" && "docs/*.md".pathChanged()
```

- 在这个版本中，您可以使用注解在 **pipelineRef:** 对象中引用远程管道。
- 在这个版本中，您可以使用 Pipelines 作为代码自动配置新的 GitHub 仓库，它会设置命名空间并为 GitHub 仓库创建 **Repository** CRD。
- 在这个版本中，Pipelines 作为代码为 **PipelineRuns** 生成指标，并带有供应商信息。
- 在这个版本中，为 **tkn-pac** 插件提供以下改进：
  - 正确检测运行的管道
  - 修复了在没有失败完成时间显示持续时间的修复
  - 在 **tkn-pac describe** 命令中显示错误片段，并突出显示错误的正则表达式模式
  - 在 **tkn-pac ls** 和 **tkn-pac describe** 命令中添加 **use-real-time** 开关
  - 导入 **tkn-pac** 日志文档

  - 在 **tkn-pac ls** 和 **tkn-pac describe** 命令中显示 **pipelineruntimeout** 作为失败。
  - 使用 **--target-pipelinerun** 选项显示特定的管道运行失败。

- 在这个版本中，您可以使用版本控制系统 (VCS) 注释或 GitHub 检查中的小片断的形式查看管道运行的错误。
- 在这个版本中，Pipelines 作为代码可以选择地检测任务中的错误（如果它们是简单格式），并在 GitHub 中将这些任务添加为注解。这个版本是开发人员预览功能的一部分。
- 在这个版本中添加了以下新命令：
  - **tkn-pac webhook add** ：在项目存储库设置中添加了一个 webhook，并在现有的 **k8s Secret** 对象中更新了 **webhook.secret** 键，而无需更新存储库。
  - **tkn-pac webhook update-token**: 更新现有 **k8s Secret** 对象的供应商令牌，而无需更新存储库。
- 这个版本增强了 **tkn-pac create repo** 命令的功能，该命令为 GitHub、GitLab 和 BitbucketCloud 创建并配置 Webhook，以及创建存储库。
- 在这个版本中，**tkn-pac describe** 命令以一定的排序顺序显示最新的五十个事件。
- 在这个版本中，在 **tkn-pac logs** 命令中添加了 **--last** 选项。
- 在这个版本中，**tkn-pac resolve** 命令会在文件模板中检测 **git_auth_secret** 时提示输入令牌。
- 在这个版本中，Pipelines 作为代码从日志片断中隐藏 secret，以避免在 GitHub 界面中公开 secret。
- 在这个版本中，为 **git_auth_secret** 自动生成 secret 是 **PipelineRun** 的所有者引用。使用 **PipelineRun** 清理 secret，而不是在管道运行执行后进行。
- 在这个版本中，增加了对使用 **/cancel** 注释取消管道运行的支持。
- 在此次更新之前，没有定义 GitHub 应用程序令牌范围，并将在每个存储库安装中使用令牌。在这个版本中，您可以使用以下参数将 GitHub 应用令牌范围到目标存储库：
  - **secret-github-app-token-scoped** ：将应用程序令牌范围到目标仓库，而不是应用程序安装可访问的每个存储库。
  - **secret-github-app-scope-extra-repos** ：使用额外的所有者或存储库自定义应用程序令牌的范围。
- 在这个版本中，您可以将 Pipelines 用作代码与在 GitLab 上托管的自己的 Git 存储库一起使用。
- 在这个版本中，您可以使用命名空间中的 kubernetes 事件的形式访问管道执行详情。这些详细信息可帮助您对管道错误进行故障排除，而无需访问 admin 命名空间。
- 在这个版本中，支持通过 Git 供应商在 Pipelines 中作为代码解析器验证 URL。
- 在这个版本中，您可以使用 **pipelines-as-code** 配置映射中的设置来设置 hub 目录的名称。
- 在这个版本中，您可以为 **max-keep-run** 参数设置最大和默认限值。
- 在这个版本中，添加了有关如何注入 Pipelines 作为代码中的自定义安全套接字层 (SSL) 证书的文档，可让您使用自定义证书连接到供应商实例。

- 在这个版本中，**PipelineRun** 资源定义包含的日志 URL 作为注解。例如， **tkn-pac describe** 命令在描述 **PipelineRun** 时显示日志链接。

- 在这个版本中，**tkn-pac** 日志显示存储库名称，而不是 **PipelineRun** 名称。

4.1.4.2. 可能会造成问题的更改

- 在这个版本中，**Conditions** 自定义资源定义(CRD) 类型已被删除。作为替代方案，请使用 **WhenExpressions** 替代。

- 在这个版本中，删除了 **tekton.dev/v1alpha1** API pipeline 资源，如 Pipeline、PipelineRun、Task、Clustertask 和 TaskRun。

- 在这个版本中，**tkn-pac setup** 命令已被删除。使用 **tkn-pac webhook add** 命令将 webhook 重新添加到现有 Git 仓库。使用 **tkn-pac webhook update-token** 命令为 Git 仓库中的现有 Secret 对象更新个人供应商访问令牌。

- 在这个版本中，运行带有默认设置的管道的命名空间不会将 **pod-security.kubernetes.io/enforce:privileged** 标签应用到工作负载。

4.1.4.3. 弃用和删除的功能

- 在 Red Hat OpenShift Pipelines 1.9.0 发行版本中，**ClusterTasks** 已被弃用，计划在以后的发行版本中删除。作为替代方案，您可以使用 **Cluster Resolver**。

- 在 Red Hat OpenShift Pipelines 1.9.0 发行版本中，在单个 **EventListener** 规格中使用 **triggers** 和 **namespaceSelector** 字段已弃用，并计划在以后的发行版本中删除。您可以在不同的 **EventListener** 规格中成功使用这些字段。

- 在 Red Hat OpenShift Pipelines 1.9.0 发行版本中，**tkn pipelinerun describe** 命令不会显示 **PipelineRun** 资源的超时。

- 在 Red Hat OpenShift Pipelines 1.9.0 发行版本中，PipelineResource' 自定义资源(CR) 已被弃用。**PipelineResource** CR 是一个技术预览功能，它是 **tekton.dev/v1alpha1** API 的一部分。

- 在 Red Hat OpenShift Pipelines 1.9.0 发行版本中，集群任务中的自定义镜像参数已弃用。另外，您可以复制集群任务并在其中使用自定义镜像。

4.1.4.4. 已知问题

- 卸载 Red Hat OpenShift Pipelines Operator 后会删除 **chains-secret** 和 **chains-config** 配置映射。当它们包含用户数据时，应保留且不删除它们。

- 当在 Windows 上运行 **tkn pac** 命令集时，您可能会收到以下出错信息： **Command finished with error: not supported by Windows。**
  临时解决方案：将 **NO_COLOR** 环境变量设置为 **true**。

- 如果 **tkn pac resolve** 命令使用已模板化的参数值，则运行 **tkn pac resolve -f <filename> | oc create -f** 命令可能无法提供预期的结果。
  临时解决方案： 要缓解这个问题，请通过运行 **tkn pac resolve -f <filename> -o tempfile.yaml** 命令将 **tkn pac resolve** 命令的输出保持在一个临时文件中，然后运行 **oc create -f tempfile.yaml** 命令。例如：**tkn pac resolve -f <filename> -o /tmp/pull-request-resolved.yaml && oc create -f /tmp/pull-request-resolved.yaml**。

4.1.4.5. 修复的问题

- 在此次更新之前，在替换空数组后，原始阵列返回一个空字符串，从而导致其中的参数无效。在这个版本中，这个问题已被解决，原始阵列返回为空。

- 在此次更新之前，如果管道运行的服务帐户中存在重复的 secret，则会导致任务 pod 创建失败。在这个版本中，这个问题已解决，即使服务帐户中存在重复的 secret，任务 pod 也会成功创建。

- 在此次更新之前，通过查看 TaskRun 的 **spec.StatusMessage** 字段，用户无法区分 **TaskRun** 是否已被用户取消或作为它一部分的 **PipelineRun**。在这个版本中，这个问题已被解决，用户可以通过查看 **TaskRun** 的 **spec.StatusMessage** 字段来区分 TaskRun 的状态。

- 在此次更新之前，在删除旧版本的无效对象时，webhook 验证会被删除。在这个版本中，这个问题已解决。

- 在此次更新之前，如果您将 **timeouts.pipeline** 参数设置为 **0**，您将无法设置 **timeouts.tasks** 参数或 **timeouts.finally** 参数。这个版本解决了这个问题。现在，当您设置 **timeouts.pipeline** 参数值时，可以设置 `timeouts.tasks` 参数或 **timeouts.finally** 参数。例如：

```
yaml
kind: PipelineRun
spec:
  timeouts:
    pipeline: "0"  # No timeout
    tasks: "0h3m0s"
```

- 在此次更新之前，如果 PipelineRun 或 TaskRun 上的另一个工具更新了标签或注解，则可能会出现竞争条件。在这个版本中，这个问题已被解决，您可以合并标签或注解。

- 在此次更新之前，日志密钥与管道控制器中的密钥不同。在这个版本中，这个问题已被解决，日志密钥已被更新，以匹配管道控制器的日志流。日志中的键已从 "ts" 改为 "timestamp"，从 "level" 改为 "severity"，从 "message" 改为 "msg"。

- 在此次更新之前，如果一个 PipelineRun 被删除并带有未知状态，则不会生成错误消息。在这个版本中，这个问题已解决，并生成错误消息。

- 在此次更新之前，若要访问 **list** 和 **push** 等捆绑包命令，需要使用 **kubeconfig** 文件。在这个版本中，这个问题已被解决，且 **kubeconfig** 文件不需要访问捆绑包命令。

- 在此次更新之前，如果在删除 TaskRuns 时运行父 PipelineRun，则将删除 TaskRuns。在这个版本中，这个问题已被解决，如果父 PipelineRun 正在运行，TaskRuns 不会被删除。

- 在此次更新之前，如果用户试图构建带有超过管道控制器的对象的捆绑包，Tekton CLI 不会显示错误消息。在这个版本中，这个问题已解决，如果用户试图构建比管道控制器中允许的限制更多的对象，Tekton CLI 会显示错误消息。

- 在此次更新之前，如果从集群中删除命名空间，Operator 不会从 **ClusterInterceptor ClusterRoleBinding** 主题中删除命名空间。在这个版本中，这个问题已被解决，Operator 会从 **ClusterInterceptor ClusterRoleBinding** 主题中删除命名空间。

- 在此次更新之前，Red Hat OpenShift Pipelines Operator 的默认安装会导致集群中的 **pipelines-scc-rolebinding 安全上下文约束** (SCC) 角色绑定资源。在这个版本中，Red Hat OpenShift Pipelines Operator 的默认安装会导致 **pipelines-scc-rolebinding 安全性上下文约束** (SCC) 角色绑定资源被从集群中移除。

- 在此次更新之前，Pipelines as Code 不会从 Pipelines as Code **ConfigMap** 对象获取更新的值。在这个版本中，这个问题已被解决，Pipelines as Code **ConfigMap** 对象会查找任何新的更改。

- 在此次更新之前，Pipelines as Code 控制器不会等待 **tekton.dev/pipeline** 标签被更新并添加 **checkrun id** 标签，这会导致竞争条件。在这个版本中，Pipelines as Code 控制器会等待 **tekton.dev/pipeline** 标签更新，然后添加 **checkrun id** 标签，这有助于避免竞争条件。

- 在此次更新之前，如果 git 仓库中已存在，**tkn-pac create repo** 命令不会覆盖 **PipelineRun**。在这个版本中，**tkn-pac create** 命令已被修复，如果 git 仓库中存在 **PipelineRun**，这会成功覆盖它。

- 在此次更新之前，**tkn pac describe** 命令不会显示每个消息的原因。在这个版本中，这个问题已被解决，**tkn pac describe** 命令会显示每个消息的原因。

- 在此次更新之前，如果使用 regex 表单的注解中的用户提供值，则拉取请求会失败，如 **refs/head/rel-***。拉取请求失败，因为它在基础分支中缺少 **refs/heads**。在这个版本中，添加了前缀，并检查它是否匹配。这解决了这个问题，拉取请求可以成功。

4.1.4.6. Red Hat OpenShift Pipelines 正式发布 1.9.1 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.9.1 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.4.7. 修复的问题

- 在此次更新之前，**tkn pac repo list** 命令不会在 Microsoft Windows 上运行。在这个版本中解决了这个问题，您可以在 Microsoft Windows 上运行 **tkn pac repo list** 命令。

- 在此次更新之前，Pipelines as Code watcher 不会接收所有配置更改事件。在这个版本中，Pipelines as Code watcher 已被更新，现在 Pipelines as Code watcher 不会丢失配置更改事件。

- 在此次更新之前，由 Pipelines as Code 创建的 pod，如 **TaskRuns** 或 **PipelineRuns** 无法访问集群中用户公开的自定义证书。在这个版本中解决了这个问题，您可以从集群中的 **TaskRuns** 或 **PipelineRuns** pod 访问自定义证书。

- 在此次更新之前，在使用 FIPS 启用的集群中，**Trigger** 资源中使用的 **tekton-triggers-core-interceptors** 内核拦截器在 Pipelines Operator 升级到 1.9 后无法正常工作。这个版本解决了这个问题。现在，OpenShift 将 MInTLS 1.2 用于其所有组件。因此，**tekton-triggers-core-interceptors** core interceptor 对 TLS 版本 1.2 及其功能进行准确运行。

- 在此次更新之前，当使用带有内部 OpenShift 镜像 registry 的管道运行时，必须在管道运行定义中硬编码到镜像的 URL。例如：

```
...
  - name: IMAGE_NAME
    value: 'image-registry.openshift-image-
registry.svc:5000/<test_namespace>/<test_pipelinerun>'
...
```

  当在 Pipelines 作为代码上下文中使用管道运行时，这些硬编码值会阻止管道运行定义在不同的集群和命名空间中使用。

  在这个版本中，您可以使用动态模板变量，而不是硬编码命名空间和管道运行名称的值，以常规化管道运行定义。例如：

```
...
  - name: IMAGE_NAME
    value: 'image-registry.openshift-image-registry.svc:5000/{{ target_namespace
```

```
}}/$(context.pipelineRun.name)'
...
```

- 在此次更新之前，Pipelines as Code 使用相同的 GitHub 令牌获取默认 GitHub 分支上同一主机上提供的远程任务。这个版本解决了这个问题。现在，Pipelines as Code 使用相同的 GitHub 令牌从任何 GitHub 分支获取远程任务。

4.1.4.8. 已知问题

- **CATALOG_REFRESH_INTERVAL** 的值(Tekton Hub CR 中使用的 Hub API **ConfigMap** 对象中的一个字段)不会更新，无法使用用户提供的自定义值进行更新。
  临时解决方案：无.您可以跟踪问题 SRVKP-2854。

4.1.4.9. 可能会造成问题的更改

- 在这个版本中，引入了一个 OLM 错误配置问题，这会阻止升级 OpenShift Container Platform。这个问题将在以后的发行版本中解决。

4.1.4.10. Red Hat OpenShift Pipelines 正式发布 1.9.2 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.9.2 包括在 OpenShift Container Platform 4.11、4.12 和 4.13 中。

4.1.4.11. 修复的问题

- 在此次更新之前，之前的版本中引入了一个 OLM 错误配置问题，这会阻止 OpenShift Container Platform 升级。在这个版本中，这个错误问题已被修复。

4.1.4.12. Red Hat OpenShift Pipelines 正式发布（GA）1.9.3 发行注记

在这个版本中，除了 4.11、4.12 和 4.13 外，Red Hat OpenShift Pipelines 正式发布 (GA) 1.9.3 包括在 OpenShift Container Platform 4.10 中。

4.1.4.13. 修复的问题

- 在这个版本中解决了大型管道的性能问题。现在，CPU 的使用率减少了 61%，内存用量减少了 44%。

- 在此次更新之前，如果某个任务因为 **when** 表达式而没有运行，则管道运行会失败。在这个版本中，通过防止在管道结果中验证跳过的任务结果解决了这个问题。现在，管道结果不会被发送，管道运行不会因为缺少结果而失败。

- 在这个版本中，为 **v1beta1** API 修复了 **pipelineref.bundle** 转换到捆绑包解析器的问题。现在，转换功能会在转换完成后将 **kind** 字段的值设置为 **Pipeline**。

- 在此次更新之前，Pipelines Operator 中的一个问题会阻止用户将 **spec.pipeline.enable-api-fields** 字段的值设置为 **beta**。在这个版本中解决了这个问题。现在，您可以在 **TektonConfig** 自定义资源中将值设为 **beta**、**alpha** 和 **stable**。

- 在此次更新之前，因为集群错误，Pipelines as Code 无法创建 secret 时，它会在 GitHub 的 check run 过程中显示临时令牌（这个过程是公开的）。在这个版本中解决了这个问题。现在，在创建 secret 失败时，这个令牌不再显示在 GitHub 检查界面中。

4.1.4.14. 已知问题

- 目前，在 OpenShift Container Platform Web 控制台中的管道运行的 **stop** 选项存在一个已知问题。**Actions** 下拉列表中的 **stop** 选项无法正常工作，它无法取消管道运行。

- 因为自定义资源定义转换失败，升级到 Pipelines 版本 1.9.x 存在一个已知问题。
临时解决方案：在升级到 Pipelines 版本 1.9.x 之前，请执行红帽客户门户网站中解决方案中提到的步骤。

4.1.5. Red Hat OpenShift Pipelines 正式发布 1.8 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.8 包括在 OpenShift Container Platform 4.10, 4.11 和 4.12 中。

4.1.5.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.8 中的新内容。

4.1.5.1.1. Pipelines

- 在这个版本中，您可以在在 ARM 硬件上运行 Red Hat OpenShift Pipelines GA 1.8 及更新的版本。这包括对 **ClusterTask** 资源和 **tkn** CLI 工具的支持。

重要

在 ARM 硬件上运行 Red Hat OpenShift Pipelines 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，为 **TaskRun** 资源实现了 **Step** 和 **Sidecar** 覆盖。

- 在这个版本中，在 **PipelineRun** 状态下添加了最小的 **TaskRun** 和 **Run** 状态。
要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。

- 在这个版本中，管道运行功能的安全终止会从 alpha 功能提升到 stable 功能。因此，之前弃用的 **PipelineRunCancelled** 状态会保持弃用，计划在以后的发行版本中被删除。
由于这个功能默认可用，所以您不再需要在 **TektonConfig** 自定义资源定义中将 **pipeline.enable-api-fields** 字段设置为 **alpha**。

- 在这个版本中，您可以使用工作区名称为管道任务指定工作区。这个更改可让您更轻松地为 **Pipeline** 和 **PipelineTask** 资源指定共享的工作区。您还可以继续显式映射工作区。
要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。

- 在这个版本中，内嵌规格中的参数会在不修改的情况下传播。

- 在这个版本中，您可以使用注解和标签指定 **PipelineRun** 资源引用的 **Task** 资源所需的元数据。这样，在管道运行过程中，依赖执行上下文的 **Task** 元数据可用。

- 在这个版本中，**params** 和 **results** 值增加了对象和字典类型。这个变化会影响向后兼容性，有时会破坏转发兼容性，比如使用较早的客户端和以后的 Red Hat OpenShift Pipelines 版本。这个版本更改了 **ArrayOrStruct** 结构，它会影响使用 Go 语言 API 作为库的项目。

- 在这个版本中，为 **PipelineRun** 状态字段的 **SkippedTasks** 字段添加了一个 **SkippingReason** 值，以便用户了解跳过给定 PipelineTask 的原因。

- 在这个版本中，支持一个 alpha 功能，您可以使用 **数组** 类型从 **Task** 对象发出结果。结果类型从**字符串**改为**ArrayOrString**。例如，任务可以指定类型来生成数组结果：

```
kind: Task
apiVersion: tekton.dev/v1beta1
metadata:
  name: write-array
  annotations:
    description: |
      A simple task that writes array
spec:
  results:
    - name: array-results
      type: array
      description: The array results
...
```

另外，您可以运行一个任务脚本来填充数组的结果：

```
$ echo -n "[\"hello\",\"world\"]" | tee $(results.array-results.path)
```

要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。

这个功能正在进行中，它是 TEP-0076 的一部分。

4.1.5.1.2. 触发器

- 在这个版本中，**EventListener** 规格中的 **TriggerGroups** 字段从 alpha 功能转换为 stable 功能。使用此字段，您可以在选择和运行一组触发器前指定一组拦截器。
由于这个功能默认可用，所以您不再需要在 **TektonConfig** 自定义资源定义中将 **pipeline.enable-api-fields** 字段设置为 **alpha**。

- 在这个版本中，**Trigger** 资源通过运行使用 HTTPS 的 **ClusterInterceptor** 服务器来支持端到端安全连接。

4.1.5.1.3. CLI

- 在这个版本中，您可以使用 **tkn taskrun export** 命令，将集群中运行的 live 任务导出到 YAML 文件，您可以使用它来将任务运行导入到另一个集群。

- 在这个版本中，您可以在 **tkn pipeline start** 命令中添加 **-o name** 标志，以便在启动后打印管道运行的名称。

- 在这个版本中，在 **tkn --help** 命令的输出中添加了可用插件列表。

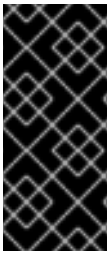
- 在这个版本中，在删除管道运行或任务运行时，您可以同时使用 **--keep** 和 **--keep-since** 标志。

- 在这个版本中，您可以使用 **Cancelled** 作为 **spec.status** 字段的值，而不是弃用的 **PipelineRunCancelled** 值。

4.1.5.1.4. Operator

- 在这个版本中，作为管理员，您可以将本地 Tekton Hub 实例配置为使用自定义数据库，而不是默认数据库。
- 在这个版本中，作为集群管理员，如果您启用本地 Tekton Hub 实例，它会定期刷新数据库，以便目录中的更改出现在 Tekton Hub web 控制台中。您可以调整刷新之间的周期。
  在以前的版本中，要将目录中的任务和管道添加到数据库中，您需要手动执行该任务或设置 cron 任务来为您完成这个任务。
- 在这个版本中，您可以使用最小配置安装并运行 Tekton Hub 实例。这样，您可以开始使用团队来确定他们可能需要的其他自定义。
- 在这个版本中，将 **GIT_SSL_CAINFO** 添加到 **git-clone** 任务，以便您可以克隆安全的存储库。

4.1.5.1.5. Tekton Chains



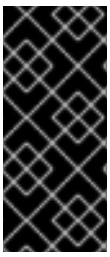
重要

Tekton 链只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，您可以使用 OIDC 而不是静态令牌登录到 vault。这个更改意味着 Spire 可以生成 OIDC 凭证，以便只允许可信工作负载登录到 vault。另外，您可以将 vault 地址作为配置值传递，而不是将其作为环境变量注入。
- 在升级 Red Hat OpenShift Pipelines Operator 后，**openshift-pipelines** 命名空间中的 Tekton Chains 的 **chains-config** 配置映射会自动重置为 default，因为在使用 Red Hat OpenShift Pipelines Operator 安装时不支持直接更新配置映射。但是，在此次更新中，您可以使用 **TektonChain** 自定义资源配置 Tekton 链。这个功能可让您在升级后保留配置，这与 **chains-config** 配置映射不同，这在升级过程中会被覆盖。

4.1.5.1.6. Tekton Hub



重要

Tekton Hub 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，如果您使用 Operator 安装一个新的 Tekton Hub 实例，则 Tekton Hub 登录默认是禁用的。要启用登录和评级功能，必须在安装 Tekton Hub 时创建 Hub API secret。

注意

因为在 Red Hat OpenShift Pipelines 1.7 中默认启用了 Tekton Hub 登录，所以如果升级 Operator，在 Red Hat OpenShift Pipelines 1.8 中会默认启用登录。要禁用此登录，请参阅从 OpenShift Pipelines 1.7.x -→ 1.8.x 升级后禁用 Tekton Hub 登录

- 在这个版本中，作为管理员，您可以将本地 Tekton Hub 实例配置为使用自定义 PostgreSQL 13 数据库，而不是默认数据库。为此，请创建一个名为 **tekton-hub-db** 的 **Secret** 资源。例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: tekton-hub-db
  labels:
    app: tekton-hub-db
type: Opaque
stringData:
  POSTGRES_HOST: <hostname>
  POSTGRES_DB: <database_name>
  POSTGRES_USER: <username>
  POSTGRES_PASSWORD: <password>
  POSTGRES_PORT: <listening_port_number>
```

- 在这个版本中，您不再需要登录到 Tekton Hub web 控制台，将目录中的资源添加到数据库。现在，当 Tekton Hub API 首次运行时，这些资源会自动添加。

- 在这个版本中，通过调用目录刷新 API 作业，每 30 分钟自动刷新目录。用户可以配置这个间隔。

4.1.5.1.7. Pipelines 作为代码（Pipelines as Code）

重要

Pipelines as Code (PAC) 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，作为开发人员，如果您试图将重复的存储库添加到 Pipelines 作为代码运行，则从 **tkn-pac** CLI 工具获得通知。输入 **tkn pac create repository** 时，每个仓库必须具有一个唯一的 URL。此通知还有助于防止劫持漏洞攻击。

- 在这个版本中，作为开发人员，您可以使用新的 **tkn-pac setup cli** 命令，使用 webhook 机制将 Git 存储库作为代码添加到 Pipelines 中。这样，即使在使用 GitHub Apps 时也可以使用 Pipelines 作为代码。此功能包括对 GitHub、GitLab 和 BitBucket 上的存储库的支持。

- 在这个版本中，Pipelines 作为代码支持 GitLab 与特征集成，如下所示：

  - 项目或组的 ACL（访问权限控制列表）
  - 允许的用户的 /**ok-to-test** 支持
  - /**retest** 支持。

- 在这个版本中，您可以使用通用表达式语言 (CEL) 执行高级管道过滤。使用 CEL，您可以使用 **PipelineRun** 资源中的注解匹配带有不同 Git 供应商事件的管道运行。例如：

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
```

```
event == "pull_request" && target_branch == "main" && source_branch == "wip"
```

- 在以前的版本中，作为开发人员，每个 Git 事件在 **.tekton** 目录中只能运行一个管道运行，如拉取请求。在这个版本中，您可以在 **.tekton** 目录中运行多个管道。Web 控制台显示运行的状态和报告。管道并行运行，并报告回 Git 提供程序接口。

- 在这个版本中，您可以通过对拉取请求为 **/test** 或 **/retest** 添加注释来测试或重新测试管道运行。您还可以按名称指定管道运行。例如，您可以输入 **/test <pipelinerun_name>** 或 **/retest <pipelinerun-name>**。

- 在这个版本中，您可以使用新的 **tkn-pac delete repository** 命令删除存储库自定义资源及其关联的 secret。

4.1.5.2. 可能会造成问题的更改

- 在这个版本中，**TaskRun** 和 **PipelineRun** 资源的默认指标级别改为以下值：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-observability
  namespace: tekton-pipelines
  labels:
    app.kubernetes.io/instance: default
    app.kubernetes.io/part-of: tekton-pipelines
data:
  _example: |
  ...
    metrics.taskrun.level: "task"
    metrics.taskrun.duration-type: "histogram"
    metrics.pipelinerun.level: "pipeline"
    metrics.pipelinerun.duration-type: "histogram"
```

- 在这个版本中，如果 **Pipeline** 和 **PipelineRun** 资源中存在注解或标签，**Run** 类型中的值将具有优先权。如果 **Task** 和 **TaskRun** 资源中存在注解或标签，则也是如此。

- 在 Red Hat OpenShift Pipelines 1.8 中，之前弃用的 **PipelineRun.Spec.ServiceAccountNames** 字段已被删除。使用 **PipelineRun.Spec.TaskRunSpecs** 字段替代。

- 在 Red Hat OpenShift Pipelines 1.8 中，之前已弃用的 **TaskRun.Status.ResourceResults.ResourceRef** 字段已被删除。使用 **TaskRun.Status.ResourceResults.ResourceName** 字段。

- 在 Red Hat OpenShift Pipelines 1.8 中，之前已弃用的 **Conditions** 资源类型已被删除。从包含它的 **Pipeline** 资源定义中删除 **Conditions** 资源。在 **PipelineRun** 定义中使用 **when** 表达式。

- 对于 Tekton 链，在这个发行版本中删除了 **tekton-provenance** 格式。通过在 **TektonChain** 自定义资源中设置 **"artifacts.taskrun.format": "in-toto"** 来使用 **in-toto** 格式。

- Red Hat OpenShift Pipelines 1.7.x 作为代码 0.5.x 附带。当前的更新附带了 Pipelines，作为代码 0.10.x。此更改在新控制器的 **openshift-pipelines** 命名空间中创建新路由。您必须在使用 Pipelines 作为代码的 GitHub Apps 或 Webhook 中更新此路由。要获取路由，请使用以下命令：

```
$ oc get route -n openshift-pipelines pipelines-as-code-controller \
  --template='https://{{ .spec.host }}'
```

- 在这个版本中，Pipelines 作为 Code 会重命名 **Repository** 自定义资源定义 (CRD) 的默认 secret 密钥。在 CRD 中，将 **token** 替换为 **provider.token**，并将 **secret** 替换为 **webhook.secret**。
- 在这个版本中，Pipelines 作为代码将特殊模板变量替换为支持用于私有仓库的多个管道运行的特殊模板变量。在管道运行中，将 **secret: pac-git-basic-auth-{{repo_owner}}-{{repo_name}}** 替换为 **secret: {{ git_auth_secret }}**。
- 在这个版本中，Pipelines as Code 更新 **tkn-pac** CLI 工具中的以下命令：
  - 将 **tkn pac repository create** 替换为 **tkn pac create repository**。
  - 将 **tkn pac repository delete** 替换为 **tkn pac delete repository**。
  - 将 **tkn pac repository list** 替换为 **tkn pac list**。

4.1.5.3. 弃用和删除的功能

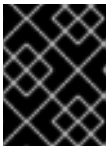
- 从 OpenShift Container Platform 4.11 开始，会删除用于安装和升级 Red Hat OpenShift Pipelines Operator 的 **preview** 和 **stable** 频道。要安装和升级 Operator，请使用适当的 **pipelines-<version>** 频道，或最新稳定版本的 **latest** 频道。例如，要安装 Pipelines Operator 版本 **1.8.x**，请使用 **pipelines-1.8** 频道。

注意

在 OpenShift Container Platform 4.10 及更早的版本中，您可以使用 **preview** 和 **stable** 频道来安装和升级 Operator。

- 对 **tekton.dev/v1alpha1** API 版本的支持（在 Red Hat OpenShift Pipelines GA 1.6 中已弃用）计划在以后的 Red Hat OpenShift Pipelines GA 1.9 发行版本中删除。
  此更改会影响管道组件，其中包括 **TaskRun**, **PipelineRun**, **Task**, **Pipeline**, 和 **tekton.dev/v1alpha1** 类似。另外，将现有资源更新为使用 **apiVersion: tekton.dev/v1beta1**，如 Migrating From Tekton v1alpha1 to Tekton v1beta1 所述。

  对 **tekton.dev/v1alpha1** API 版本的程序错误修复和支持仅到当前 GA 1.8 生命周期结束为止。



重要

对于 Tekton Operator，**operator.tekton.dev/v1alpha1** API 版本**没有**弃用。您不需要更改此值。

- 在 Red Hat OpenShift Pipelines 1.8 中，**PipelineResource** 自定义资源 (CR) 可用，但不再被支持。**PipelineResource** CR 是一个技术预览功能，它属于 **tekton.dev/v1alpha1** API 的一部分，它已被弃用，并计划在以后的 Red Hat OpenShift Pipelines GA 1.9 发行版本中删除。
- 在 Red Hat OpenShift Pipelines 1.8 中，**Condition** 自定义资源 (CR) 已删除。**Condition** CR 是 **tekton.dev/v1alpha1** API 的一部分，它已被弃用，计划在以后的 Red Hat OpenShift Pipelines GA 1.9 发行版本中删除。
- 在 Red Hat OpenShift Pipelines 1.8 中，**gsutil** 的 **gcr.io** 镜像已被删除。这个删除可能会破坏带有依赖于此镜像的 **Pipeline** 资源的集群。程序错误修复和支持只到 Red Hat OpenShift Pipelines 1.7 生命周期结束为主。
- 在 Red Hat OpenShift Pipelines 1.8 中，**Pipeline.Status.TaskRuns** 和 **PipelineRun.Status.Runs** 字段已弃用，计划在以后的发行版本中被删除。请参阅 PipelineRuns 中的 TEP-0100: 嵌入式 TaskRuns 和 Runs Status。

- 在 Red Hat OpenShift Pipelines 1.8 中，**pipelineRunCancelled** 状态已弃用，计划在以后的发行版本中被删除。现在，**PipelineRun** 对象的安全终止已从 alpha 功能提升到 stable 功能。（请参阅 TEP-0058: Graceful Pipeline Run Termination。）另外，您可以使用 **Cancelled** 状态替换 **pipelineRunCancelled** 状态。
  您不需要更改 **Pipeline** 和 **Task** 资源。如果您有会取消管道运行的工具，则必须在下一个发行版本中更新这些工具。这个更改也会影响 CLI、IDE 扩展等工具，以便支持新的 **PipelineRun** 状态。

  由于这个功能默认可用，所以您不再需要在 **TektonConfig** 自定义资源定义中将 **pipeline.enable-api-fields** 字段设置为 **alpha**。

- 在 Red Hat OpenShift Pipelines 1.8 中，**PipelineRun** 中的 **timeout** 字段已弃用。反之，使用 **PipelineRun.Timeouts** 字段，它现在从 alpha 功能提升到 stable 功能。
  由于这个功能默认可用，所以您不再需要在 **TektonConfig** 自定义资源定义中将 **pipeline.enable-api-fields** 字段设置为 **alpha**。

- 在 Red Hat OpenShift Pipelines 1.8 中，**init** 容器在 **LimitRange** 对象的默认请求计算中省略。

4.1.5.4. 已知问题

- **s2i-nodejs** 管道无法使用 **nodejs:14-ubi8-minimal** 镜像流来执行 source-to-image (S2I) 构建。使用该镜像流会生成一个 **error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127** 信息。
  临时解决方案： 使用 **nodejs:14-ubi8** 而不是 **nodejs:14-ubi8-minimal** 镜像流。

- 当您运行 Maven 和 Jib-Maven 集群任务时，默认容器镜像仅支持 Intel(x86)架构。因此，在 ARM、IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE (s390x) 集群上的任务将失败。
  作为临时解决方案，您可以通过将 **MAVEN_IMAGE** 参数值设置为 **maven:3.6.3-adoptopenjdk-11** 来指定自定义镜像。

  提示

  在您使用 **tkn hub** 在 ARM、IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE(s390x) 上基于 Tekton Catalog 安装任务前，请验证是否可以在这些平台上执行任务。要检查 **ppc64le** 和 **s390x** 是否列在任务信息的"Platforms"部分，您可以运行以下命令： **tkn hub info task <name>**

- 在 ARM、IBM Power Systems、IBM Z 和 LinuxONE 中，不支持 **s2i-dotnet** 集群任务。

- 隐式参数映射会错误地将参数从顶级 **Pipeline** 或 **PipelineRun** 定义传递给 **taskRef** 任务。映射应该只在顶层资源中使用任务进行，并带有非线 **taskSpec** 规格。这个问题只通过将 **TektonConfig** 自定义资源定义的 **pipeline** 部分中的 **enable-api-fields** 字段设置为 **alpha** 来影响启用了这个功能的集群。

4.1.5.5. 修复的问题

- 在此次更新之前，管道的指标在 Web 控制台的 Developer 视图中运行的指标不完整且过期。在这个版本中，这个问题已被解决，指标正确。

- 在此次更新之前，如果管道有两个失败并有 **retries=2** 的并行任务，则最终任务永不会运行，管道超时并且无法运行。例如，**pipelines-operator-subscription** 任务会间歇性地失败，并显示以下错误消息： **Unable to connect to server: EOF**。在这个版本中，这个问题已被解决，以便最终任务始终运行。

- 在此次更新之前，如果管道运行因为任务运行失败而停止，则其他任务运行可能无法完成重试。因此，不会调度 **finally** 任务，这会导致管道挂起。这个版本解决了这个问题。**TaskRuns** 和 **Run** 对象可以在管道运行停止时重试，即使是安全停止也是如此，以便管道运行可以完成。

- 在这个版本中，当 **TaskRun** 对象的命名空间中存在一个或多个 **LimitRange** 对象时，会改变如何计算资源要求。现在，当从 **LimitRange** 对象发出请求时，调度程序现在考虑 **step** 容器并排除所有其他应用容器，如 sidecar 容器。

- 在此次更新之前，在特定的条件下，标志软件包可能会错误地解析子命令，并带有双短划线终止符 **--**。在这种情况下，它会运行 entrypoint 子命令，而不是实际命令。在这个版本中解决了这个问题，这个问题会解决，entrypoint 会运行正确的命令。

- 在此次更新之前，如果拉取镜像失败，控制器可能会生成多个 panics，或者其拉取 (pull) 状态不完整。在这个版本中解决了这个问题，方法是检查 **step.ImageID** 值而不是 **status.TaskSpec** 值。

- 在此次更新之前，取消一个包含未计划的自定义任务的管道运行会生成 **PipelineRunCouldntCancel** 错误。在这个版本中解决了这个问题。您可以取消包含非调度自定义任务的管道运行，而不会生成这个错误。

- 在此次更新之前，如果在 **$params["<NAME>"]** 或 **$params['<NAME>']** 的 **<NAME>** 中包括了一个句点字符 (.)，则不会提取句点右侧的任何部分。例如，对于 **$params["org.ipsum.lorem"]**，只会提取y **org** 部分。
  在这个版本中解决了这个问题，**$params** 可以提取完整的值。例如，**$params["org.ipsum.lorem"]** 和 **$params['org.ipsum.lorem']** 是有效的，**<NAME>** 的完整值 **org.ipsum.lorem** 都会被提取。

  如果 **<NAME>** 没有包含在单引号或双引号中，还会抛出一个错误。例如，**$params.org.ipsum.lorem** 是无效，并生成验证错误。

- 在这个版本中，**Trigger** 资源支持自定义拦截器，并确保自定义拦截器服务的端口与 **ClusterInterceptor** 定义文件中的端口相同。

- 在此次更新之前，Tekton Chains 和 Operator 组件的 **tkn version** 命令无法正常工作。在这个版本中解决了这个问题，命令可以正常工作，并返回这些组件的版本信息。

- 在此次更新之前，如果您运行 **tkn pr delete --ignore-running** 命令，且管道运行没有 **status.condition** 值，**tkn** CLI 工具会生成 null-pointer 错误 (NPE)。在这个版本中解决了这个问题，CLI 工具现在会生成错误，并正确忽略仍然运行的管道运行。

- 在此次更新之前，如果您使用 **tkn pr delete --keep <value>** 或 **tkn tr delete --keep <value>** 命令，且管道运行或任务运行的数量小于其中的值，则命令不会如预期返回一个错误。在这个版本中解决了这个问题，以便命令在这些条件下正确返回错误。

- 在此次更新之前，如果您使用 **tkn pr delete** 或 **tkn tr delete** 命令以及 **-p** 或 **-t** 标志与 **--ignore-running** 标志一起，则命令会错误地删除运行或待处理资源。在这个版本中解决了这个问题，这些命令可以正确地忽略运行或待处理资源。

- 在此版本中，您可以使用 **TektonChain** 自定义资源配置 Tekton 链。这个功能可让您在升级后保留配置，这与 **chains-config** 配置映射不同，这在升级过程中会被覆盖。

- 在这个版本中，除了 **buildah** 和 **s2i** 集群任务外，**ClusterTask** 资源不再默认以 root 用户身份运行。

- 在此次更新之前，当将 **init** 用作第一个参数并后跟两个或更多个参数时，Red Hat OpenShift Pipelines 1.7.1 上的任务会失败。在这个版本中，这些标志会被正确解析，任务可以运行成功。

- 在此次更新之前，因为存在无效的角色绑定，在 OpenShift Container Platform 4.9 和 4.10 上安装 Red Hat OpenShift Pipelines Operator 会失败，并显示以下错误消息：

  ```
  error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
  RoleBinding.rbac.authorization.k8s.io
  "openshift-operators-prometheus-k8s-read-binding" is invalid:
  roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io", Kind:"Role",
  Name:"openshift-operator-read"}: cannot change roleRef
  ```

  在这个版本中解决了这个问题，不再会发生失败。

- 在以前的版本中，升级 Red Hat OpenShift Pipelines Operator 会导致重新创建 **pipeline** 服务帐户，这意味着链接到服务帐户的 secret 会丢失。在这个版本中解决了这个问题。在升级过程中，Operator 不再重新创建 **pipeline** 服务帐户。因此，已附加到 **pipeline** 服务帐户的 secret 在升级后会保留，资源（任务和管道）可以继续正常工作。

- 在这个版本中，如果在 **TektonConfig** 自定义资源 (CR) 中配置了基础架构节点设置，Pipelines as Code pod 在基础架构节点上运行。

- 在以前的版本中，使用资源修剪器时，每个命名空间 Operator 创建了一个在独立容器中运行的命令。这个设计在有大量命名空间的集群中消耗太多资源。例如，要运行单个命令，带有 1000 个命名空间的集群会在 pod 中生成 1000 个容器。
  在这个版本中解决了这个问题。它将基于命名空间的配置传递给作业，以便所有命令在循环中的一个容器中运行。

- 在 Tekton Chains 中，您必须定义一个名为 **signed-secrets** 的 secret，以存放用于签名任务和镜像的密钥。但是，在更新前，更新 Red Hat OpenShift Pipelines Operator 会重置或覆盖这个 secret，密钥会丢失。在这个版本中解决了这个问题。现在，如果在通过 Operator 安装 Tekton Chains 后配置了 secret，secret 会保留，且升级不会覆盖它。

- 在此次更新之前，所有 S2I 构建任务都会失败，并显示类似以下消息的错误：

  ```
  Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
  operation not permitted
  time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
  /proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
  time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
  ```

  在这个版本中，**pipeline-scc** 安全性上下文约束 (SCC) 与 **Buildah** 和 **S2I** 集群任务所需的 **SETFCAP** 功能兼容。因此，**Buildah** 和 **S2I** 构建任务可以成功运行。

  要成功运行 **Buildah** 集群任务和 **S2I** 构建任务，适用于使用各种语言和框架编写的应用程序，添加以下适当的 **steps** 对象（如 **build** 和 **push**）的代码片段。

  ```
  securityContext:
    capabilities:
      add: ["SETFCAP"]
  ```

- 在此次更新之前，安装 Red Hat OpenShift Pipelines Operator 所需的时间比预期的要长。这个版本会优化一些设置来加快安装过程。

- 在这个版本中，Buildah 和 S2I 集群任务比之前的版本要少。有些步骤已合并到一个步骤中，它们最好使用 **ResourceQuota** 和 **LimitRange** 对象，且不需要更多资源。

- 在这个版本中，在集群任务中升级 Buildah、**tkn** CLI 工具和 **skopeo** CLI 工具版本。

- 在此次更新之前，如果任何命名空间处于 **Terminating** 状态，Operator 会在创建 RBAC 资源时失败。在这个版本中，Operator 忽略了命名空间处于 **Terminating** 状态，并创建 RBAC 资源。

- 在此次更新之前，prune cronjobs 的 pod 没有在基础架构节点上调度。相反，它们被调度到 worker 节点上，或没有调度到任何节点。在这个版本中，如果在 **TektonConfig** 自定义资源 (CR) 中进行了配置，这些类型的 pod 可以在基础架构节点上调度。

4.1.5.6. Red Hat OpenShift Pipelines 正式发布 1.8.1 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.8.1 包括在 OpenShift Container Platform 4.10, 4.11 和 4.12 中。

4.1.5.6.1. 已知问题

- 默认情况下，容器具有 restricted 权限来提高安全性。restricted 权限适用于 Red Hat OpenShift Pipelines Operator 中的所有控制器 pod，以及一些集群任务。由于权限受限，**git-clone** 集群任务在特定配置下会失败。
  临时解决方案：无.您可以跟踪此问题 SRVKP-2634。

- 当安装程序集合处于失败状态时，**TektonConfig** 自定义资源的状态会被错误地显示为 **True**，而不是 **False**。

  示例：失败的安装程序集

```
$ oc get tektoninstallerset
NAME                                READY   REASON
addon-clustertasks-nx5xz            False   Error
addon-communityclustertasks-cfb2p   True
addon-consolecli-ftrb8              True
addon-openshift-67dj2               True
addon-pac-cf7pz                     True
addon-pipelines-fvllm               True
addon-triggers-b2wtt                True
addon-versioned-clustertasks-1-8-hqhnw   False   Error
pipeline-w75ww                      True
postpipeline-lrs22                  True
prepipeline-ldlhw                   True
rhosp-rbac-4dmgb                    True
trigger-hfg64                       True
validating-mutating-webhoook-28rf7   True
```

  示例： 不正确的 TektonConfig 状态

```
$ oc get tektonconfig config
NAME     VERSION   READY   REASON
config   1.8.1     True
```

4.1.5.6.2. 修复的问题

- 在此次更新之前，pruner 会删除运行管道的任务运行，并显示以下警告：**some tasks were indicated completed without ancestors being done**。在这个版本中，pruner 会保留作为运行管道一部分的任务运行。

- 在此次更新之前，**pipeline-1.8** 是安装 Red Hat OpenShift Pipelines Operator 1.8.x 的默认频道。在这个版本中，**latest** 是默认频道。
- 在此次更新之前，作为 Code 控制器 pod 的 Pipelines 无法访问用户公开的证书。在这个版本中，Pipelines as Code 可以访问路由，Git 存储库由自签名或自定义证书保护。
- 在此次更新之前，在从 Red Hat OpenShift Pipelines 1.7.2 升级到 1.8.0 后，任务会失败并带有 RBAC 错误。在这个版本中，任务会成功运行，不会出现 RBAC 错误。
- 在此次更新之前，无法使用 **tkn** CLI 工具删除包含类型为 **array** 的 **result** 对象的任务运行和管道运行。在这个版本中，可以使用 **tkn** CLI 工具删除包含类型为 **array** 的 **result** 对象的任务运行和管道运行。
- 在此次更新之前，如果管道规格包含带有类型为 **array** 的 **ENV_VARS** 参数的任务，则管道运行会失败，并带有以下错误：**invalid input params for task func-buildpacks: param types don't match the user-specified type: [ENV_VARS]**。在这个版本中，带有这样的管道和任务规格的管道不会失败。
- 在此次更新之前，集群管理员无法向 **Buildah** 集群任务提供 **config.json** 文件来访问容器 registry。在这个版本中，集群管理员可以使用 **dockerconfig** 工作区为 **Buildah** 集群任务提供 **config.json** 文件。

4.1.5.7. Red Hat OpenShift Pipelines General Availability 1.8.2 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA) 1.8.2 包括在 OpenShift Container Platform 4.10, 4.11 和 4.12 中。

4.1.5.7.1. 修复的问题

- 在此次更新之前，当使用 SSH 密钥克隆存储库时，**git-clone** 任务会失败。在这个版本中，**git-init** 任务中的非 root 用户的角色会被删除，SSH 程序会在 **$HOME/.ssh/** 目录中查找正确的密钥。

4.1.6. Red Hat OpenShift Pipelines 正式发行 1.7 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA)1.7 包括在 OpenShift Container Platform 4.9, 4.10, 和 4.11 中。

4.1.6.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.7 中的新内容。

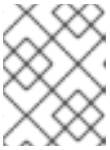
4.1.6.1.1. Pipelines

- 在这个版本中，**pipelines-<version>** 是安装 Red Hat OpenShift Pipelines Operator 的默认频道。例如，安装 Pipelines Operator **1.7** 的默认频道是 **pipelines-1.7**。集群管理员还可使用 **latest** 频道来安装 Operator 的最新稳定版本。

注意

preview 和 **stable** 频道将在以后的版本中被弃用并删除。

- 当您在用户命名空间中运行命令时，您的容器以 **root** 身份运行（用户 ID **0**），但对主机具有用户特权。在这个版本中，要在用户命名空间中运行 pod，您必须传递 CRI-O 期望的注解。
  - 要为所有用户添加这些注解，请运行 **oc edit clustertask buildah** 命令并编辑 **buildah** 集群任务。
  - 要将注解添加到特定命名空间，请将集群任务导出为该命名空间的任务。
- 在此次更新之前，如果没有满足某些条件，则 **when** 表达式会跳过一个 **Task** 对象及其依赖的任务。在这个版本中，您可以限制 **when** 表达式只保护 **Task** 对象，而不是它的依赖项任务。要启用此更新，请在 **TektonConfig** CRD 中将 **scope-when-expressions-to-task** 标志设置为 **true**。



注意

scope-when-expressions-to-task 标记已弃用，并将在以后的发行版本中删除。作为 Pipelines 的最佳实践，请只使用 **when** 表达式范围到被保证的 **Task**。

- 在这个版本中，您可以在任务中的工作区的 **subPath** 字段中使用变量替换。
- 在这个版本中，您可以使用带单引号或双引号的 bracket 表示法引用参数和结果。在此次更新之前，只能使用点表示法。例如，现在以下内容等同于：
  - **$(param.myparam)**, **$(param['myparam'])**, and **$(param["myparam"])**. 您可以使用单引号或双引号括起包含有问题的字符的参数名称，如 **"."**。例如，**$(param['my.param'])** 和 **$(param["my.param"])**。
- 在这个版本中，您可以在任务定义中包含步骤的 **onError** 参数，而无需启用 **enable-api-fields** 标志。

4.1.6.1.2. 触发器

- 在这个版本中，**feature-flag-triggers** 配置映射具有一个新的字段 **labels-exclusion-pattern**。您可以将此字段的值设置为正则表达式(regex)模式。控制器过滤出与事件监听器从事件监听器传播到为事件监听器创建的资源匹配的正则表达式模式的标签。
- 在这个版本中，**TriggerGroups** 字段添加到 **EventListener** 规格中。使用此字段，您可以在选择和运行一组触发器前指定要运行的拦截器。要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。
- 在这个版本中，**Trigger** 资源支持 **TriggerTemplate** 模板定义的自定义运行。
- 在这个版本中，Triggers 支持从 **EventListener** pod 发送 Kubernetes 事件。
- 在这个版本中，以下对象提供了计数指标：**ClusterInteceptor**、**EventListener**、**TriggerTemplate**、**ClusterTriggerBinding** 和 **TriggerBinding**。
- 在这个版本中，**ServicePort** 规格添加到 Kubernetes 资源。您可以使用此规格来修改公开事件监听器服务的端口。默认端口为 **8080**。
- 在这个版本中，您可以使用 **EventListener** 规格中的 **targetURI** 字段在触发器处理过程中发送云事件。要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。
- 在这个版本中，**tekton-triggers-eventlistener-roles** 对象除了已存在的 **create** 动词外，现在还添加了一个 **patch** 动词。

- 在这个版本中，**securityContext.runAsUser** 参数会从事件监听器部署中删除。

4.1.6.1.3. CLI

- 在这个版本中，**tkn [pipeline | pipelinerun] export** 命令会导出一个管道或管道作为 YAML 文件运行。例如：
  - 在 **openshift-pipelines** 命名空间中导出名为 **test_pipeline** 的管道：

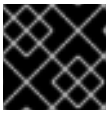
```
$ tkn pipeline export test_pipeline -n openshift-pipelines
```

  - 在 **openshift-pipelines** 命名空间中导出名为 **test_pipeline_run** 的管道运行：

```
$ tkn pipelinerun export test_pipeline_run -n openshift-pipelines
```

- 在这个版本中，**--grace** 选项添加到 **tkn pipelinerun cancel** 中。使用 **--grace** 选项终止管道运行，而不是强制终止。要启用此功能，在 **TektonConfig** 自定义资源定义的 **pipeline** 部分，您必须将 **enable-api-fields** 字段设置为 **alpha**。

- 在这个版本中，在 **tkn version** 命令的输出中添加了 Operator 和链版本。



重要

Tekton 链是技术预览功能。

- 在这个版本中，当取消管道运行时，**tkn pipelinerun describe** 命令显示所有取消的任务运行。在此修复之前，仅显示一个任务运行。

- 在这个版本中，您可以在运行 **tkn [t | p | p | ct] start** 命令时跳过使用 **--skip-optional-workspace** 标记时，为可选工作区提供请求规格。您也可以在以交互模式运行时跳过它。

- 在这个版本中，您可以使用 **tkn chain** 命令管理 Tekton 链。您还可以使用 **--chains-namespace** 选项指定要安装 Tekton 链的命名空间。

重要

Tekton 链是技术预览功能。

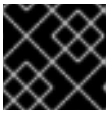
4.1.6.1.4. Operator

- 在这个版本中，您可以使用 Red Hat OpenShift Pipelines Operator 来安装和部署 Tekton Hub 和 Tekton 链。

重要

集群中的 Tekton Hub 链和部署是技术预览功能。

- 在这个版本中，您可以查找并使用 Pipelines 作为代码(PAC)作为附加选项。



重要

作为代码，管道是技术预览功能。

- 在这个版本中，您可以通过将 **communityClusterTasks** 参数设置为 **false** 来禁用社区集群任务的安装。例如：

```
...
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "true"
    - name: communityClusterTasks
      value: "false"
...
```

- 在这个版本中，您可以通过 **Developer** 视角禁用 Tekton Hub 集成，方法是将 **TektonConfig** 自定义资源中的 **enable-devconsole-integration** 标志设置为 **false**。例如：

```
...
hub:
  params:
    - name: enable-devconsole-integration
      value: "true"
...
```

- 在这个版本中，**operator-config.yaml** 配置映射启用 **tkn version** 命令的输出来显示 Operator 版本。
- 在这个版本中，**argocd-task-sync-and-wait** 任务的版本被修改为 **v0.2**。
- 对于这个版本的 **TektonConfig** CRD， **oc get tektonconfig** 命令会显示 OPerator 版本。
- 在这个版本中，服务监控器添加到 Triggers 指标中。

4.1.6.1.5. Hub

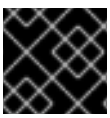
重要

在集群中部署 Tekton Hub 是一个技术预览功能。

Tekton Hub 可帮助您发现、搜索和共享 CI/CD 工作流可重复使用的任务和管道。Tekton Hub 的一个公共实例位于 hub.tekton.dev 中。

集群管理员可以使用 Red Hat OpenShift Pipelines 1.7 推出，集群管理员也可以在企业集群中安装并部署 Tekton Hub 的自定义实例。您可以使用特定于您的机构的可重复使用的任务和管道来策展目录。

4.1.6.1.6. 链



重要

Tekton 链是技术预览功能。

Tekton Chains 是一个 Kubernetes 自定义资源定义(CRD)控制器。您可以使用它来管理使用 Red Hat OpenShift Pipelines 创建的任务和管道的供应链安全。

默认情况下，Tekton Chains 会监控 OpenShift Container Platform 集群中运行的任务。链会对完成的任务运行进行快照，将其转换为一个或多个标准有效负载格式，并签署并存储所有工件。

Tekton Chains 支持以下功能：

- 您可以使用加密密钥类型和服务（如 **cosign** ）为任务运行、任务运行结果和 OCI registry 镜像签名。
- 您可以使用"测试"格式，如 **in-toto**。
- 您可以使用 OCI 存储库作为存储后端安全存储签名和签名工件。

4.1.6.1.7. 管道作为代码(PAC)

重要

作为代码，管道是技术预览功能。

使用 Pipelines 作为 Code，具有所需权限的集群管理员和用户可以将管道模板定义为源代码 Git 存储库的一部分。当由源代码推送或配置的 Git 存储库的拉取请求触发时，该功能将运行管道和报告状态。

作为代码的管道支持以下功能：

- 拉取请求状态。在迭代拉取请求时，拉取请求的状态和控制会在托管 Git 存储库的平台上进行。
- GitHub 检查 API 来设置管道运行的状态，包括重新检查。
- GitHub 拉取请求和提交事件。
- 在注释中拉取请求操作，如 **/retest**。
- Git 事件过滤，以及每个事件的独立管道。
- Pipelines 中的自动任务解析用于本地任务、Tekton Hub 和远程 URL。
- 使用 GitHub blob 和对象 API 来检索配置。
- 通过 GitHub 机构或使用 Prow 风格的 **OWNER** 文件访问控制列表(ACL)
- **tkn** CLI 工具的 **tkn pac** 插件，您可以使用它们管理 Pipelines 作为代码软件仓库和 bootstrap。
- 支持 GitHub 应用程序、GitHub Webhook、Bitbucket 服务器和 Bitbucket 云。

4.1.6.2. 已弃用的功能

- 中断更改：此更新从 **TektonConfig** 自定义资源(CR)中删除 **disable-working-directory-overwrite** 和 **disable-home-env-overwrite** 字段。因此， **TektonConfig** CR 不再自动设置 **$HOME** 环境变量和 **workingDir** 参数。您仍然可以使用 **Task** 自定义资源定义(CRD)中的 **env** 和 **workingDir** 字段来设置 **$HOME** 环境变量和 **workingDir** 参数。
- **Conditions** 自定义资源定义(CRD)类型已弃用，计划在以后的发行版本中删除。反之，使用推荐的 **when** 表达式。

- 有问题的更改： **Triggers** 资源验证模板，并在没有指定 **EventListener** 和 **TriggerBinding** 值时生成错误。

4.1.6.3. 已知问题

- 当您运行 Maven 和 Jib-Maven 集群任务时，默认容器镜像仅支持 Intel(x86)架构。因此，在 ARM、IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE (s390x) 集群上的任务将失败。作为临时解决方案，您可以通过将 **MAVEN_IMAGE** 参数值设置为 **maven:3.6.3-adoptopenjdk-11** 来指定自定义镜像。

  提示

  在您使用 **tkn hub** 在 ARM、IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE(s390x) 上基于 Tekton Catalog 安装任务前，请验证是否可以在这些平台上执行任务。要检查 **ppc64le** 和 **s390x** 是否列在任务信息的"Platforms"部分，您可以运行以下命令： **tkn hub info task <name>**

- 在 IBM Power Systems、IBM Z 和 LinuxONE 中，不支持 **s2i-dotnet** 集群任务。

- 您无法使用 **nodejs:14-ubi8-minimal** 镜像流，因为这样做会生成以下错误：

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

- 隐式参数映射会错误地将参数从顶级 **Pipeline** 或 **PipelineRun** 定义传递给 **taskRef** 任务。映射应该只在顶层资源中使用任务进行，并带有非线 **taskSpec** 规格。这个问题只通过将 **TektonConfig** 自定义资源定义的 **pipeline** 部分中的 **enable-api-fields** 字段设置为 **alpha** 来影响启用了这个功能的集群。

4.1.6.4. 修复的问题

- 在这个版本中，如果 **Pipeline** 和 **PipelineRun** 对象定义中存在 **labels** 和 **annotations** 的元数据，**PipelineRun** 类型中的值会具有优先权。您可以观察 **Task** 和 **TaskRun** 对象的类似行为。

- 在这个版本中，如果 **timeout.tasks** 字段或 **timeout.finally** 字段设置为 **0**，则 **timeout.pipeline** 也被设置为 **0**。

- 在这个版本中，**-x** set 标记已从不使用 shebang 的脚本中删除。在这个版本中，减少了脚本执行的潜在数据泄漏。

- 在这个版本中，Git 凭证中的用户名中存在的任何反斜杠字符都会被转义，并带有 **.gitconfig** 文件中的额外反斜杠。

- 在这个版本中，**EventListener** 对象的 **finalizer** 属性不需要清理日志记录和配置映射。

- 在这个版本中，与事件监听器服务器关联的默认 HTTP 客户端会被删除，并添加自定义 HTTP 客户端。因此，超时有所改进。

- 在这个版本中，Triggers 集群角色现在可以处理所有者引用。

- 在这个版本中，当多个拦截器返回扩展时，事件监听器中的竞争条件不会发生。

- 在这个版本中，**tkn pr delete** 命令不会删除使用 **ignore-running** 标记运行的管道。
- 在这个版本中，当您修改任何附加组件参数时，Operator pod 不会继续重启。
- 在这个版本中，如果订阅和配置自定义资源中没有配置，**tkn serve** CLI pod 会被调度到 infra 节点上。
- 在这个版本中，在升级过程中不会删除指定版本的集群任务。

4.1.6.5. Red Hat OpenShift Pipelines 正式发行 1.7.1 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA)1.7.1 包括在 OpenShift Container Platform 4.9, 4.10, 和 4.11 中。

4.1.6.5.1. 修复的问题

- 在此次更新之前，升级 Red Hat OpenShift Pipelines Operator 会删除与 Tekton Hub 关联的数据库中的数据并安装新的数据库。在这个版本中，Operator 升级会保留数据。
- 在此次更新之前，只有集群管理员才能访问 OpenShift Container Platform 控制台中的管道指标。在这个版本中，与其他集群角色的用户也可以访问管道指标。
- 在此次更新之前，管道对包含发出大型终止消息的任务的管道运行失败。管道运行失败，因为 pod 中所有容器的终止信息总数不能超过 12 KB。在这个版本中，使用同一镜像的 **place-tools** 和 **step-init** 初始化容器会合并，以减少每个任务的 pod 中运行的容器数量。该解决方案可减少运行失败管道的几率，这可尽量减少在任务 pod 中运行的容器数量。但是，它不会删除终止消息的最大允许大小限制。
- 在此次更新之前，直接从 Tekton Hub web 控制台访问资源 URL 会导致 Nginx **404** 错误。在这个版本中，Tekton Hub web 控制台镜像已被修复，以便直接从 Tekton Hub web 控制台访问资源 URL。
- 在此次更新之前，对于每个命名空间，资源修剪器作业都会创建一个单独的容器来修剪资源。在这个版本中，资源修剪器作业作为一个容器中的一个循环运行所有命名空间的命令。

4.1.6.6. Red Hat OpenShift Pipelines 正式发布 1.7.2 发行注记

在这个版本中，Red Hat OpenShift Pipelines General Availability (GA)1.7.2 包括在 OpenShift Container Platform 4.9、4.10 及以后的版本中。

4.1.6.6.1. 已知问题

- 在升级 Red Hat OpenShift Pipelines Operator 后，**openshift-pipelines** 命名空间中的 Tekton Chains 的 **chains-config** 配置映射会自动重置为默认。目前，这个问题还没有临时解决方案。

4.1.6.6.2. 修复的问题

- 在此次更新之前，当将 **init** 用作第一个参数并后跟两个或更多个参数时，Pipelines 1.7.1 上的任务会失败。在这个版本中，这些标志会被正确解析，任务可以运行成功。
- 在此次更新之前，因为存在无效的角色绑定，在 OpenShift Container Platform 4.9 和 4.10 上安装 Red Hat OpenShift Pipelines Operator 会失败，并显示以下错误消息：

```
error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
RoleBinding.rbac.authorization.k8s.io "openshift-operators-prometheus-k8s-read-binding" is
```

```
invalid: roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io",
Kind:"Role", Name:"openshift-operator-read"}: cannot change roleRef
```

在这个版本中，Red Hat OpenShift Pipelines Operator 安装有不同的角色绑定命名空间，以避免与其他 Operator 安装冲突。

- 在此次更新之前，升级 Operator 会触发 Tekton Chains 的 **signed-secrets** secret 键重置为默认值。在这个版本中，在升级 Operator 后自定义 secret 密钥会保留。

注意

升级到 Red Hat OpenShift Pipelines 1.7.2 会重置密钥。但是，当您升级到将来的版本时，会需要保留该密钥。

- 在此次更新之前，所有 S2I 构建任务都会失败，并显示类似以下消息的错误：

```
Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
operation not permitted
time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
/proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
```

在这个版本中，**pipeline-scc** 安全性上下文约束 (SCC) 与 **Buildah** 和 **S2I** 集群任务所需的 **SETFCAP** 功能兼容。因此，**Buildah** 和 **S2I** 构建任务可以成功运行。

要成功运行 **Buildah** 集群任务和 **S2I** 构建任务，适用于使用各种语言和框架编写的应用程序，添加以下适当的 **steps** 对象（如 **build** 和 **push**）的代码片段。

```
securityContext:
  capabilities:
    add: ["SETFCAP"]
```

4.1.6.7. Red Hat OpenShift Pipelines 正式发布 1.7.3 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA)1.7.3 包括在 OpenShift Container Platform 4.9, 4.10, 和 4.11 中。

4.1.6.7.1. 修复的问题

- 在此次更新之前，如果任何命名空间处于 **Terminating** 状态，Operator 会在创建 RBAC 资源时失败。在这个版本中，Operator 忽略了命名空间处于 **Terminating** 状态，并创建 RBAC 资源。

- 在以前的版本中，升级 Red Hat OpenShift Pipelines Operator 会导致重新创建 **pipeline** 服务帐户，这意味着链接到服务帐户的 secret 会丢失。在这个版本中解决了这个问题。在升级过程中，Operator 不再重新创建 **pipeline** 服务帐户。因此，已附加到 **pipeline** 服务帐户的 secret 在升级后会保留，资源（任务和管道）可以继续正常工作。

4.1.7. Red Hat OpenShift Pipelines 正式发行 1.6 发行注记

在这个版本中，Red Hat OpenShift Pipelines 正式发行(GA)1.6 包括在 OpenShift Container Platform 4.9 中。

4.1.7.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.6 中的新内容。

- 在这个版本中，您可以配置管道或任务的 **start** 命令（使用 **--output <string>**，其中 **<string>** 是 **yaml** 或 **json** ）来返回 YAML 或 JSON 格式的字符串。如果没有使用 **--output** 选项，**start** 命令会返回人类友好的消息，而其他程序很难解析该消息。返回 YAML 或 JSON 格式的字符串对于持续集成(CI)环境非常有用。例如，在创建了资源后，您可以使用 **yq** 或 **jq** 解析有关资源的 YAML 或 JSON 格式消息，并等待该资源终止而不使用 **showlog** 选项。

- 在这个版本中，您可以使用 Podman 的 **auth.json** 身份验证文件对 registry 进行身份验证。例如，您可以使用 **tkn bundle push** 来使用 Podman 而不是 Docker CLI 推送到远程 registry。

- 在这个版本中，如果您使用 **tkn [taskrun | pipelinerun] delete --all** 命令，您可以使用新的 **--keep-since <minutes>** 选项来指定一个分钟数，存在时间少于这个分钟数的运行会被保留。例如，若要保持存在的时间少于五分钟的运行，输入 **tkn [taskrun | pipelinerun] delete -all --keep-since 5**。

- 在这个版本中，当删除任务运行或管道运行时，您可以将 **--parent-resource** 和 **--keep-since** 选项一起使用。例如，**tkn pipelinerun delete --pipeline pipelinename --keep-since 5** 命令会保留其父资源名为 **pipelinename**，其存在时间为五分钟或更短的管道运行。对于任务运行，**tkn tr delete -t <taskname> --keep-since 5** 和 **tkn tr delete --clustertask <taskname> --keep-since 5** 有类似效果。

- 在这个版本中，增加了对用于 **v1beta1** 资源的触发器资源的支持。

- 在这个版本中，在 **tkn pipelinerun delete** 和 **tkn taskrun delete** 命令中添加了一个 **ignore-running** 选项。

- 在这个版本中，为 **tkn task** 和 **tkn clustertask** 命令添加一个 **create** 子命令。

- 在这个版本中，当使用 **tkn pipelinerun delete --all** 命令时，您可以使用新的 **--label <string>** 选项来根据标签过滤管道运行。另外，您还可以在 **--label** 选项中使用 **=** 和 **==** 作为相等运算符，或者 **!=** 作为不相等运算符。例如，**tkn pipelinerun delete --all --label asdf** 和 **tkn pipelinerun delete --all --label==asdf** 命令都会删除带有 **asdf** 标签的所有管道运行。

- 在这个版本中，您可以从配置映射获取已安装的 Tekton 组件版本，或者从部署控制器获取配置映射的版本。

- 在这个版本中，触发器支持使用 **feature-flags** 和 **config-defaults** 配置映射来分别配置功能标记和设置默认值。

- 在这个版本中，添加了一个新的指标 **eventlistener_event_count**，您可以使用它来统计 **EventListener** 资源接收的事件。

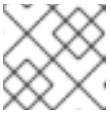
- 在这个版本中添加了 **v1beta1** Go API 类型。在这个版本中，触发器支持 **v1beta1** API 版本。在当前发行版本中，**v1alpha1** 功能已弃用，并将在以后的发行版本中删除。改为使用 **v1beta1** 功能替代。

- 在当前发行版本中，默认启用自动运行资源。另外，您可以使用以下新注解为每个命名空间单独配置自动运行任务运行和管道运行：

  - **operator.tekton.dev/prune.schedule**：如果此注解的值与 **TektonConfig** 自定义资源定义中指定的值不同，则会在该命名空间中创建新的 cron 作业。

  - **operator.tekton.dev/prune.skip**：设置为 **true** 时，配置它的命名空间不会被修剪。

- **operator.tekton.dev/prune.resources**：此注解接受以逗号分隔的资源列表。要修剪单一资源，如管道运行，将此注解设置为 **"pipelinerun"**。要修剪多个资源，如任务运行和管道运行，将此注解设置为 **"taskrun, pipelinerun"**。
- **operator.tekton.dev/prune.keep**：使用此注解保留资源而不进行修剪。
- **operator.tekton.dev/prune.keep-since**：使用此注解来根据其存在的时间来保留资源。此注解的值必须等于资源的年龄（以分钟为单位）。例如，若要保留在五天内创建的资源，请将 **keep-since** 设置为 **7200**。



注意

keep 和 **keep-since** 注释是互斥的。对于任何资源，您只能配置其中一个。

- **operator.tekton.dev/prune.strategy**：将此注解的值设置为 **keep** 或 **keep-since**。

- 管理员可以禁用在整个集群中创建 **pipeline** 服务帐户创建，从而防止因为错误地使用关联的 SCC（这与 **anyuid** 非常相似）造成权限升级的问题。
- 现在，您可以使用 **TektonConfig** 自定义资源(CR)和单个组件的 CR 配置功能标记和组件，如 **TektonPipeline** 和 **TektonTriggers**。此级别的颗粒度有助于为各个组件的 Tekton OCI 捆绑包自定义和测试 alpha 功能。
- 现在，您可以为 **PipelineRun** 资源配置可选的 **Timeouts** 字段。例如，您可以单独为管道运行、每个任务运行以及 **finally** 任务配置超时。
- **TaskRun** 资源生成的 Pod 现在会设置 pod 的 **activeDeadlineSeconds** 字段。这可让 OpenShift 将它们视为终止，并允许您将具有特定作用域的 **ResourceQuota** 对象用于容器集。
- 您可以使用 configmaps 在任务运行、管道运行、任务和管道上消除指标标签或标签类型。另外，您可以为测量持续时间配置不同类型的指标，如直方图、量表或最后一个值。
- 您可以统一定义 pod 的请求和限值，因为 Tekton 现在通过考虑 **Min**、**Max**、**Default** 和 **DefaultRequest** 字段来全面支持 **LimitRange** 对象。
- 引入了以下 alpha 功能：
  - 现在，管道运行可以在运行 **finally** 任务后停止，而不是之前停止直接运行所有任务的行为。在这个版本中添加了以下 **spec.status** 值：
    - **StoppedRunFinally** 将在当前运行的任务完成后停止，然后运行 **finally** 任务。
    - **CancelledRunFinally** 将立即取消正在运行的任务，然后运行 **finally** 任务。
    - **Cancelled** 将保留 **PipelineRunCancelled** 状态提供的以前行为。

注意

Cancelled 状态替换已弃用的 **PipelineRunCancelled** 状态，该状态将在 **v1** 版本中删除。

  - 现在，您可以使用 **oc debug** 命令将任务运行设置为 debug 模式，这会暂停执行并允许您检查 pod 中的特定步骤。

- 当您将一个步骤的 **onError** 字段设置为 **continue** 时，会记录该步骤的退出代码并传递给后续步骤。但是，任务运行不会失败，任务中其余步骤的执行将继续。要保留现有的行为，您可以将 **onError** 字段的值设置为 **stopAndFail**。
- 任务现在可以接受比实际使用更多的参数。启用 alpha 功能标记时，参数可以隐式传播到内联规格。例如，内联任务可以访问其父管道运行的参数，而不必为任务明确定义每个参数。
- 如果您为 alpha 功能启用标记，则 **When** 表达式下的条件将仅适用于直接关联的任务，而不是任务的依赖项。要将 **When** 表达式应用到关联的任务及其依赖关系中，您必须将表达式与每个相依任务单独关联。请注意，进入下一步是 Tekton 的任何新 API 版本中 **When** 表达式的默认行为。现有的默认行为将被弃用，而由此更新替代。

- 当前发行版本允许您通过在 **TektonConfig** 自定义资源(CR)中指定 **nodeSelector** 和 **tolerations** 值来配置节点选择。Operator 将这些值添加到它创建的所有部署中。
  - 要为 Operator 的控制器和 webhook 部署配置节点选择，请在安装 Operator 后编辑 **Subscription** CR 规格中的 **config.nodeSelector** 和 **config.tolerations** 字段。
  - 要在基础架构节点上部署 OpenShift Pipelines 的其余 control plane pod，请使用 **nodeSelector** 和 **tolerations** 字段更新 **TektonConfig** CR。然后，修改会应用到 Operator 创建的所有 pod。

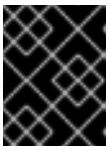
4.1.7.2. 已弃用的功能

- 在 CLI 0.21.0 中，已弃用了对 **clustertask**, **task**, **taskrun**, **pipeline**, 和 **pipelinerun** 的所有 **v1alpha1** 资源的支持。这些资源现已过时，并将在以后的发行版本中删除。
- 在 Tekton Triggers v0.16.0 中，冗余的 **status** 标签已从 **EventListener** 资源的指标中删除。

重要

中断更改： **status** 标签已从 **eventlistener_http_duration_seconds_*** 指标中删除。删除基于 **status** 标签的查询。

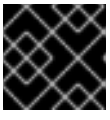
- 在当前发行版本中，**v1alpha1** 功能已弃用，并将在以后的发行版本中删除。在这个版本中，您可以改为使用 **v1beta1** Go API 类型。触发器现在支持 **v1beta1** API 版本。
- 在当前发行版本中，**EventListener** 资源会在触发器完成处理前发送响应。



重要

中断更改：使用此更改时，**EventListener** 资源会在创建资源时停止响应 **201 Created** 的状态代码。相反，它使用 **202 Accepted** 的响应代码进行响应。

- 当前发行版本从 **EventListener** 资源中删除 **podTemplate** 字段。



重要

中断更改：删除了作为 #1100 一部分的 **podTemplate** 字段。

- 当前发行版本从 **EventListener** 资源规格中删除已弃用的 **replicas** 字段。

重要

中断更改：已弃用的 **replicas** 字段已被删除。

- 在 Red Hat OpenShift Pipelines 1.6 中，**HOME="/tekton/home"** 和 **workingDir="/workspace"** 的值从 **Step** 对象的规格中删除。
  相反，Red Hat OpenShift Pipelines 将 **HOME** 和 **workingDir** 设置为运行 **Step** 对象的容器定义的值。您可以在 **Step** 对象的规格中覆盖这些值。

  要使用旧的行为，您可以将 **TektonConfig** CR 中的 **disable-working-directory-overwrite** 和 **disable-home-env-overwrite** 字段改为 **false** ：

```
apiVersion: operator.tekton.dev/v1alpha1
 kind: TektonConfig
 metadata:
   name: config
 spec:
   pipeline:
     disable-working-directory-overwrite: false
     disable-home-env-overwrite: false
 ...
```

重要

TektonConfig CR 中的 **disable-working-directory-overwrite** 和 **disable-home-env-overwrite** 字段现已弃用，并将在以后的发行版本中删除。

4.1.7.3. 已知问题

- 当您运行 Maven 和 Jib-Maven 集群任务时，默认容器镜像仅支持 Intel(x86)架构。因此，在 IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE(s390x)集群上的任务将失败。作为临时解决方案，您可以通过将 **MAVEN_IMAGE** 参数值设置为 **maven:3.6.3-adoptopenjdk-11** 来指定自定义镜像。

- 在 IBM Power Systems、IBM Z 和 LinuxONE 中，不支持 **s2i-dotnet** 集群任务。

- 在您使用 **tkn hub** 在 IBM Power Systems(ppc64le)、IBM Z 和 LinuxONE(s390x) 上基于 Tekton Catalog 安装任务前，请验证是否可以在这些平台上执行任务。要检查 **ppc64le** 和 **s390x** 是否列在任务信息的"Platforms"部分，您可以运行以下命令： **tkn hub info task <name>**

- 您无法使用 **nodejs:14-ubi8-minimal** 镜像流，因为这样做会生成以下错误：

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

4.1.7.4. 修复的问题

- 现在，IBM Power Systems、IBM Z 和 LinuxONE 支持 **tkn hub** 命令。

- 在更新前，当用户运行 **tkn** 命令并完成管道运行后，终端不可用，即使指定了 **retries** 也是如此。在任务运行或管道运行中指定超时无效。在这个版本中解决了这个问题，终端可以在运行命令后可用。

- 在更新前，运行 **tkn pipelinerun delete --all** 会删除所有资源。在这个版本中，处于 running 状态的资源无法被删除。

- 在这个版本中，使用 **tkn version --component=<component>** 命令不会返回组件版本。在这个版本中解决了这个问题，这个命令会返回组件版本。

- 在更新前，当您使用 **tkn pr logs** 命令时，它会以错误的任务顺序显示管道输出日志。在这个版本中解决了这个问题，已完成的 **PipelineRuns** 的日志会在适当的 **TaskRun** 执行顺序中列出。

- 在这个版本中，编辑正在运行的管道规格可能会阻止管道运行在完成后停止。在这个版本中，只获取定义一次，然后使用存储在状态中的规格进行验证，从而解决了这个问题。当 **PipelineRun** 或 **TaskRun** 指代的 **Pipeline** 或 **Task** 在运行时会发生变化，这个变化减少了出现竞争条件的可能性。

- **When** 表达式值可以有数组参数引用，例如 **values: [$(params.arrayParam[*])]**。

4.1.7.5. Red Hat OpenShift Pipelines 正式发行版本 1.6.1 发行注记

4.1.7.5.1. 已知问题

- 从旧版本升级到 Red Hat OpenShift Pipelines 1.6.1 后，Pipelines 可能会进入不一致的状态，因为您无法对 Tekton 资源（tasks 和 pipelines）执行任何操作(create/delete/apply)。例如，在删除资源时，您可能会遇到以下错误：

  ```
  Error from server (InternalError): Internal error occurred: failed calling webhook
  "validation.webhook.pipeline.tekton.dev": Post "https://tekton-pipelines-webhook.openshift-
  pipelines.svc:443/resource-validation?timeout=10s": service "tekton-pipelines-webhook" not
  found.
  ```

4.1.7.5.2. 修复的问题

- Red Hat OpenShift Pipelines 设置的 **SSL_CERT_DIR** 环境变量(**/tekton-custom-certs**)不会用证书文件覆盖以下默认系统目录：

  - **/etc/pki/tls/certs**
  - **/etc/ssl/certs**
  - **/system/etc/security/cacerts**

- Horizontal Pod Autoscaler 可以管理 Red Hat OpenShift Pipelines Operator 控制的部署副本数。在这个发行版本中，如果最终用户或集群代理更改了计数，Red Hat OpenShift Pipelines Operator 不会重置管理的部署的副本计数。但是，升级 Red Hat OpenShift Pipelines Operator 时会重置副本。

- 现在，根据 **TektonConfig** 自定义资源中指定的节点选择器和容限(toleration)限制将把 **tkn** CLI 的 pod 调度到节点上。

4.1.7.6. Red Hat OpenShift Pipelines 正式发行版本 1.6.2 发行注记

4.1.7.6.1. 已知问题

- 当您创建新项目时，**pipeline** 服务帐户的创建会被延迟，并删除现有集群任务和管道模板需要超过 10 分钟。

4.1.7.6.2. 修复的问题

- 在此次更新之前，在从旧版本升级到 Red Hat OpenShift Pipelines 1.6.1 后，会为管道创建多个 Tekton 安装程序集实例。在这个版本中，Operator 可确保升级后每个类型为 **TektonInstallerSet** 的一个实例都存在。

- 在此次更新之前，Operator 中的所有协调程序都使用组件版本在升级到旧版本的 Red Hat OpenShift Pipelines 1.6.1 期间决定资源重新创建。因此，这些资源不会被重新创建，其组件版本没有在升级过程中改变。在这个版本中，Operator 使用 Operator 版本而不是组件版本来决定升级过程中的资源重新创建。

- 在此次更新之前，升级后，集群中缺少管道 Webhook 服务。这是因为配置映射上的升级死锁。在这个版本中，如果集群中不存在配置映射，添加了一个机制来禁用 Webhook 验证。因此，在升级后管道 Webhook 服务会保留在集群中。

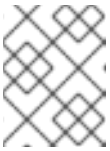
- 在此次更新之前，在任何配置更改到命名空间后，自动运行操作的 cron 作业会被重新创建。在这个版本中，只有在命名空间中有相关注解改变时，才会重新创建 auto-pruning 的 cron 任务。

- Tekton Pipelines 的上游版本被修订为 **v0.28.3**，它包括以下修复：
  - 修复 **PipelineRun** 或 **TaskRun** 对象，以允许进行标签或注解。
  - 对于隐式参数：
    - 不要将 **PipelineSpec** 参数应用到 **TaskRefs** 对象。
    - 禁用 **Pipeline** 对象的隐式参数行为。

4.1.7.7. Red Hat OpenShift Pipelines 正式发布 1.6.3 发行注记

4.1.7.7.1. 修复的问题

- 在此次更新之前，Red Hat OpenShift Pipelines Operator 从组件（如 Pipelines 和 Triggers）中安装 Pod 安全策略。但是，作为组件的一部分提供的 Pod 安全策略在以前的版本中已被弃用。在这个版本中，Operator 会停止从组件安装 Pod 安全策略。因此，以下升级路径会受到影响：
  - 从 Pipelines 1.6.1 或 1.6.2 升级到 Pipelines 1.6.3 会删除 Pod 安全策略，包括 Pipelines 和 Triggers 组件中的 Pod 安全策略。
  - 从 Pipelines 1.5.x 升级到 1.6.3 会保留从组件安装的 Pod 安全策略。作为集群管理员，您可以手动删除它们。



注意

当您升级到将来的版本时，Red Hat OpenShift Pipelines Operator 会自动删除所有过时的 Pod 安全策略。

- 在此次更新之前，只有集群管理员才能访问 OpenShift Container Platform 控制台中的管道指标。在这个版本中，与其他集群角色的用户也可以访问管道指标。

- 在此次更新之前，带有 Pipelines Operator 的基于角色的访问控制(RBAC)问题会导致升级或安装组件的问题。这个版本提高了安装各种 Red Hat OpenShift Pipelines 组件的可靠性和一致性。

- 在此次更新之前，在 **TektonConfig** CR 中将 **clusterTasks** 和 **pipelineTemplates** 字段设置为 **false** 时，会降低删除集群任务和管道模板。这个版本提高了 Tekton 资源（如集群任务和管道模板）的生命周期管理速度。

4.1.7.8. Red Hat OpenShift Pipelines 正式发布 1.6.4 发行注记

4.1.7.8.1. 已知问题

- 从 Red Hat OpenShift Pipelines 1.5.2 升级到 1.6.4 后，访问事件监听程序路由会返回 **503** 错误。临时解决方案：修改事件监听器路由的 YAML 文件中的目标端口。

1. 为相关命名空间提取路由名称。

```
$ oc get route -n <namespace>
```

2. 编辑路由以修改 **targetPort** 字段的值。

```
$ oc edit route -n <namespace> <el-route_name>
```

示例：现有事件监听程序路由

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
  port:
    targetPort: 8000
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

示例：修改事件监听程序路由

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
  port:
    targetPort: http-listener
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

4.1.7.8.2. 修复的问题

- 在此次更新之前，如果任何命名空间处于 **Terminating** 状态，Operator 会在创建 RBAC 资源时失败。在这个版本中，Operator 忽略了命名空间处于 **Terminating** 状态，并创建 RBAC 资源。

- 在此次更新之前，该任务会在没有注解指定关联的 Tekton 控制器发行版本而运行失败或重启。在这个版本中，包含适当的注解会被自动运行，没有失败或重启的任务。

4.1.8. Red Hat OpenShift Pipelines 正式发行 1.5 发行注记

Red Hat OpenShift Pipelines General Availability (GA) 1.5 现在包括在 OpenShift Container Platform 4.8 中。

4.1.8.1. 兼容性和支持列表

这个版本中的一些功能当前还只是一个技术预览。它们并不适用于在生产环境中使用。

在下表中，被标记为以下状态的功能：

TP	技术预览
GA	公开发行

请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

表 4.2. 兼容性和支持列表

功能	版本	支持状态
Pipelines	0.24	GA
CLI	0.19	GA
目录	0.24	GA
触发器	0.14	TP
Pipeline 资源	-	TP

如果您有疑问或希望提供反馈信息，请向产品团队发送邮件 pipelines-interest@redhat.com。

4.1.8.2. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.5 中的新内容。

- Pipeline 运行和任务运行将由目标命名空间中的 cron 作业自动修剪。cron 作业使用 **IMAGE_JOB_PRUNER_TKN** 环境变量来获取 **tkn image** 的值。在这个版本中，**TektonConfig** 自定义资源包括以下字段：

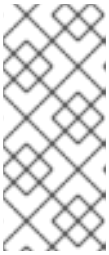
```
...
pruner:
  resources:
    - pipelinerun
    - taskrun
```

```
  schedule: "*/5 * * * *" # cron schedule
  keep: 2 # delete all keeping n
...
```

- 在 OpenShift Container Platform 中，您可以通过修改 **TektonConfig** 自定义资源中的新参数 **clusterTasks** 和 **pipelinesTemplates** 的值来自定义 Tekton Add-ons 组件的安装：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "true"
...
```

如果您使用 **TektonConfig** 创建附加组件，或者直接使用 Tekton Add-ons 创建附加组件，则允许进行自定义。但是，如果没有传递参数，控制器会添加带有默认值的参数。



注意

- 如果使用 **TektonConfig** 自定义资源创建附加组件，并且稍后在 **Addon** 自定义资源中更改参数值，则 **TektonConfig** 自定义资源中的值将覆盖更改。
- 只有在 **clusterTasks** 参数的值为 **true** 时，才可将 **pipelinesTemplates** 参数的值设置为 **true**。

- **enableMetrics** 参数添加到 **TektonConfig** 自定义资源中。您可以使用它来禁用服务监控器，这是 OpenShift Container Platform 的 Tekton Pipelines 的一部分。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  pipeline:
    params:
    - name: enableMetrics
      value: "true"
...
```

- 添加了 EventListener OpenCensus 指标（在进程级别捕获指标）。
- 触发器现在具有标签选择器；您可以使用标签为事件监听程序配置触发器。
- 添加了用于注册拦截器的 **ClusterInterceptor** 自定义资源定义，允许您注册您可以插入的新 **Interceptor** 类型。另外，还会进行以下相关更改：

    - 在触发器规格中，您可以使用包含 **ref** 字段的新 API 来配置拦截器来引用集群拦截器。另外，您可以使用 **params** 字段添加传递到拦截器进行处理的参数。
    - 捆绑的拦截器 CEL、GitHub、GitLab 和 BitBucket 已迁移。它们使用新的 **ClusterInterceptor** 自定义资源定义来实施。
    - 核心拦截器迁移到新格式，使用旧语法创建的任何新触发器都会自动切换到新的 **ref** 或 **params** 语法。

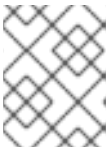
- 要在显示日志时禁用任务名称的前缀或步骤，可将 **--prefix** 选项用于 **log** 命令。
- 要显示特定组件的版本，请使用 **tkn version** 命令中的新的 **--component** 标志。
- 添加了 **tkn hub check-upgrade** 命令，其他命令会根据管道版本进行修改。此外，**search** 命令输出中也会显示目录名称。
- 在 **start** 命令中添加了对可选工作区的支持。
- 如果 **plugins** 目录中不存在插件，则会在当前路径中搜索它们。
- **tkn start [task | clustertask | pipeline]** 命令以互动方式启动，并询问 **params** 值，即使指定了默认参数也是如此。若要停止交互式提示，可在调用 命令时传递 **--use-param-defaults** 标志。例如：

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-api \
    -p git-url=https://github.com/openshift/pipelines-vote-api.git \
    -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api \
    --use-param-defaults
```

- **version** 字段在 **tkn task describe** 命令中添加。
- 如果只有一个资源，则会自动选择 **TriggerTemplate** 或 **TriggerBinding** 或 **ClusterTriggerBinding** 或 **Eventlistener** 等资源的选项会添加到 **describe** 命令中。
- 在 **tkn pr describe** 命令中添加了跳过的任务部分。
- 添加了对 **tkn clustertask logs** 的支持。
- 移除 **config.yaml** 中的 YAML 合并和变量。另外，现在 **kustomize** 和 **ytt** 等工具可以更轻松地使用 **release.yaml** 文件。
- 添加了对包含句点字符(".")的资源名称的支持。
- **PodTemplate** 规格中的 **hostAliases** 数组添加到主机名解析的 pod 级别覆盖中。它通过修改 **/etc/hosts** 文件来实现。
- 引入了一个变量 **$(tasks.status)**，用于访问任务的聚合执行状态。
- 为 Windows 添加入口点二进制构建。

4.1.8.3. 已弃用的功能

- 在 **when** 表达式中，对写入的字段的支持会被删除。**when** 表达式仅支持使用小写字母编写的字段。



注意

如果您在 Tekton Pipelines **v0.16** (Operator **v1.2.x**) 中应用了一个带有 **when** 表达式的管道，则必须重新应用它。

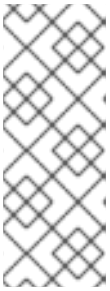
- 当您将 Red Hat OpenShift Pipelines Operator 升级到 **v1.5** 时，**openshift-client** 和 **openshift-client-v-1-5-0** 集群任务会具有 **SCRIPT** 参数。但是，**ARGS** 参数和 **git** 资源会从 **openshift-client** 集群任务的规格中删除。这个变化可能会造成问题，只有在 **ClusterTask** 资源的 **name** 字段中没有特定版本的集群任务才可以无缝地进行升级。
要防止管道运行出现问题，在升级后使用 **SCRIPT** 参数，因为它会将之前在 **ARGS** 参数中指定的值移到集群任务的 **SCRIPT** 参数中。例如：

```
...
- name: deploy
  params:
  - name: SCRIPT
    value: oc rollout status <deployment-name>
  runAfter:
    - build
  taskRef:
    kind: ClusterTask
    name: openshift-client
...
```

- 当您从 Red Hat OpenShift Pipelines Operator **v1.4** 升级到 **v1.5** 时，**TektonConfig** 自定义资源的配置集名称现在会改变。

表 4.3. TektonConfig 自定义资源的配置集

Pipelines 1.5 中的配置集	Pipelines 1.4 中的对应的配置集	安装的 Tekton 组件
All (*默认配置集*)	All (*默认配置集*)	Pipelines、Triggers、Add-ons
Basic	默认	Pipelines、Triggers
Lite	Basic	Pipelines

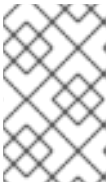


注意

如果在 **TektonConfig** 自定义资源的 **config** 实例中使用 **profile: all**，则不需要更改资源规格。

但是，如果安装的 Operator 在升级前是 Default 或 Basic 配置集，则必须在升级后编辑 **TektonConfig** 自定义资源的 **config** 实例。例如，如果配置在升级前是 **profile: basic**，请确保在升级到 Pipelines 1.5 后为 **profile: lite**。

- **disable-home-env-overwrite** 和 **disable-working-dir-overwrite** 字段现已弃用，并将在以后的发行版本中删除。在本发行版本中，这些标志的默认值被设置为 **true**，以便向后兼容。



注意

在下一个发行版本 (Red Hat OpenShift Pipelines 1.6) 中，**HOME** 环境变量不会自动设置为 **/tekton/home**，对于任务运行的默认工作目录也不会设置为 **/workspace**。这些默认设置与步骤中镜像 Dockerfile 设置的任何值冲突。

- **ServiceType** 和 **podTemplate** 字段已从 **EventListener** spec 中删除。
- 控制器服务帐户不再请求集群范围的权限来列出和监视命名空间。
- **EventListener** 资源的状态有一个名为 **Ready** 的新条件。

注意

将来，**EventListener** 资源的其他状态条件将会被弃用，而使用 **Ready** 状态条件。

- **EventListener** 响应中的 **eventListener** 和 **namespace** 字段已弃用。现在改为使用 **eventListenerUID** 字段。
- **replicas** 在 **EventListener** spec 中已被弃用。相反，**spec.replicas** 字段移到 **KubernetesResource** spec 中的 **spec.resources.kubernetesResource.replicas**。

注意

replicas 字段将在以后的发行版本中删除。

- 配置内核拦截器的旧方法已弃用。但是，它会继续正常工作，直到在以后的版本中被删除。现在，**Trigger** 资源中的拦截器被配置为使用新的基于 **ref** 和 **params** 的语法。生成的默认 webhook 会自动将旧语法的使用切换为新触发器的新语法。
- 对于 **ClusterRoleBinding** 资源，使用 **Userbac.authorization.k8s.io/v1** 而不是已弃用的 **deprecatedrbac.authorization.k8s.io/v1beta1**。
- 在集群角色中，对资源（如 **serviceaccounts**、**secrets**、**configmaps** 和 **limitranges** 等）的集群范围的写访问权限已被移除。另外，对资源（如 **deployments**、**statefulsets** 和 **deployment/finalizers** 等）的集群范围的访问权限也被移除。
- Tekton 不再使用 **caching.internal.knative.dev** 组中的 **image** 自定义资源定义，它已在此发行版本中排除。

4.1.8.4. 已知问题

- git-cli 集群任务使用 alpine/git 基础镜像构建，它预期 **/root** 作为用户的主目录。但是，这并没有在 **git-cli** 集群任务中显式设置。
  在 Tekton 中，任务的每个步骤都会使用 **/tekton/home** 覆盖默认主目录，除非另有指定。这会覆盖基础镜像的 **$HOME** 环境变量，从而导致 **git-cli** 集群任务失败。

  此问题有望在即将发布的版本中解决。对于 Red Hat OpenShift Pipelines 1.5 及更早的版本，您可以*使用以下任一临时解决方案*来避免 **git-cli** 集群任务失败：

  - 在步骤中设置 **$HOME** 环境变量，使其不会被覆盖。

    1. [可选] 如果您使用 Operator 安装了 Red Hat OpenShift Pipelines，然后将 **git-cli** 集群任务克隆到一个单独的任务中。此方法可确保 Operator 不会覆盖对集群任务所做的更改。

2. 执行 **oc edit clustertasks git-cli** 命令。

3. 将预期的 **HOME** 环境变量添加到步骤的 YAML 中：

```
...
steps:
  - name: git
    env:
    - name: HOME
      value: /root
    image: $(params.BASE_IMAGE)
    workingDir: $(workspaces.source.path)
...
```

警告

对于 Operator 安装的 Red Hat OpenShift Pipelines，如果您在更改 **HOME** 环境变量前没有将 **git-cli** 集群任务克隆到单独的任务中，则会在 Operator 协调过程中覆盖更改。

- 禁用在 **feature-flags** 配置映射中覆盖 **HOME** 环境变量。

  1. 执行 **oc edit -n openshift-pipelines configmap feature-flags** 命令。

  2. 将 **disable-home-env-overwrite** 标志的值设置为 **true**。

警告

- 如果使用 Operator 安装 Red Hat OpenShift Pipelines，则在 Operator 协调过程中会覆盖更改。
- 修改 **disable-home-env-overwrite** 标志的默认值可能会破坏其他任务和集群任务，因为它会更改所有任务的默认值。

- 为 **git-cli** 集群任务使用其他服务帐户，因为在使用管道的默认服务帐户时会覆盖 **HOME** 环境变量。

  1. 创建新服务帐户。

  2. 将您的 Git secret 链接到您刚才创建的服务帐户。

  3. 在执行任务或管道时使用服务帐户。

- 在 IBM Power Systems、IBM Z 和 LinuxONE 中，不支持 **s2i-dotnet** 集群任务和 **tkn hub** 命令。
- 当您运行 Maven 和 Jib-Maven 集群任务时，默认容器镜像仅支持 Intel(x86)架构。因此，在 IBM

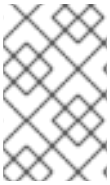
Power Systems(ppc64le)、IBM Z 和 LinuxONE(s390x)集群上的任务将失败。作为临时解决方案，您可以通过将 **MAVEN_IMAGE** 参数值设置为 **maven:3.6.3-adoptopenjdk-11** 来指定自定义镜像。

4.1.8.5. 修复的问题

- **dag** 任务中的 **when** 表达式不允许指定任何其他任务访问执行状态的上下文变量 （**$(tasks.<pipelineTask>.status)**）。
- 使用 Owner UID 而不是 Owner 名称，这有助于在 **PipelineRun** 资源快速删除并重新创建的情况下，避免通过删除 **volumeClaimTemplate** PVC 创建的竞争条件。
- 为非 root 用户触发的 **build-base** 镜像添加了 **pullrequest-init** 的新 Dockerfile。
- 当一个管道或任务在运行时使用 **-f** 选项，且它的定义中的 **param** 中没有定义 **type**，将生成一个验证错误，而不是管道或任务以静默方式运行失败。
- 对于 **tkn start [task | pipeline | clustertask]** 命令，**--workspace** 标记的描述现在一致。
- 在解析参数时，如果遇到空数组，则对应的交互式帮助现在会显示为空字符串。

4.1.9. Red Hat OpenShift Pipelines 正式发布 1.4 发行注记

Red Hat OpenShift Pipelines General Availability（GA）1.4 现在包括在 OpenShift Container Platform 4.7 中。



注意

除了 stable 和 preview Operator 频道外，Red Hat OpenShift Pipelines Operator 1.4.0 还带有 ocp-4.6、ocp-4.5 和 ocp-4.4 弃用的频道。这些过时的频道及其支持将在以下 Red Hat OpenShift Pipelines 发行版本中删除。

4.1.9.1. 兼容性和支持列表

这个版本中的一些功能当前还只是一个技术预览。它们并不适用于在生产环境中使用。

在下表中，被标记为以下状态的功能：

TP	技术预览
GA	公开发行

请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

表 4.4. 兼容性和支持列表

功能	版本	支持状态
Pipelines	0.22	GA
CLI	0.17	GA

功能	版本	支持状态
目录	0.22	GA
触发器	0.12	TP
Pipeline 资源	-	TP

如果您有疑问或希望提供反馈信息，请向产品团队发送邮件 pipelines-interest@redhat.com。

4.1.9.2. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.4 中的新内容。

- 自定义任务有以下改进：
  - Pipeline 结果现在可以引用自定义任务生成的结果。
  - 自定义任务现在可以使用工作区、服务帐户和 pod 模板来构建更复杂的自定义任务。
- **finally** 的任务包括以下改进：
  - 在 **finally** 任务中支持 **when** 表达式，它可以有效地保证执行，并提高了任务的可重复使用性。
  - **finally** 任务可以被配置为消耗同一管道中任何任务的结果。

注意

OpenShift Container Platform 4.7 web 控制台中不支持 **when** 表达式和 **finally** 任务。

- 添加了对 **dockercfg** 或 **dockerconfigjson** 类型的多个 secret 的支持，以便在运行时进行身份验证。
- 添加支持 **git-clone** 任务的 sparse-checkout 的功能。这可让您将存储库的子集克隆为本地副本，并帮助您限制克隆的存储库的大小。
- 您可以创建管道以待处理状态运行，而无需实际启动它们。在负载非常重的集群中，这允许 Operator 控制管道运行的开始时间。
- 确保为控制器手动设置 **SYSTEM_NAMESPACE** 环境变量；这在之前被默认设置。
- 现在，一个非 root 用户被添加到管道的构建基础镜像中，以便 **git-init** 能够以非 root 用户身份克隆存储库。
- 支持在管道运行启动前在已解析的资源间验证依赖项。管道中的所有结果变量都必须有效，管道中的可选工作区只能传递给期望管道启动运行的任务。
- controller 和 Webhook 作为非 root 组运行，并且删除了它们的多余功能，以使它们更加安全。
- 您可以使用 **tkn pr logs** 命令查看重试任务运行的日志流。

- 您可以使用 **tkn tr delete** 命令中的 **--clustertask** 选项删除与特定集群任务关联的所有任务。
- 通过引入一个新的 **customResource** 字段，增加了对在 **EventListener** 资源中使用 Knative 服务的支持。
- 当事件有效负载没有使用 JSON 格式时，会显示错误消息。
- 源控制拦截器（如 GitLab、BitBucket 和 GitHub）现在使用新的 **InterceptorRequest** 或 **InterceptorResponse** 类型的接口。
- 新的 CEL 功能 **marsagingJSON** 被实现，以便您可以将 JSON 对象或数组编码到字符串。
- 为 CEL 添加了 HTTP 处理器，并添加了源控制内核拦截器。它将四个核心拦截器打包到单一 HTTP 服务器中，该服务器部署在 **tekton-pipelines** 命名空间中。**EventListener** 对象通过 HTTP 服务器将事件转发到拦截器。每个拦截器都位于不同的路径。例如，CEL 拦截器位于 **/cel** 路径中。
- **pipelines-scc** 安全性上下文约束（SCC）与默认的 **pipeline** 服务帐户一同使用。此新服务帐户与 **anyuid** 类似，但 OpenShift Container Platform 4.7 的 SCC 在 YAML 中定义中有一个小的差别：

```
fsGroup:
  type: MustRunAs
```

4.1.9.3. 已弃用的功能

- 不支持 pipeline 资源存储中的 **build-gcs** 子类型和 **gcs-fetcher** 镜像。
- 在集群任务的 **TaskRun** 字段中，删除标签 **tekton.dev/task**。
- 对于 webhook，移除了与字段 **admissionReviewVersions** 对应的 **v1beta1** 值。
- 用于构建和部署的 **creds-init** 帮助程序镜像已被删除。
- 在触发器 spec 和绑定中，弃用的字段 **template.name** 已被删除，并使用 **template.ref** 替代。您应该更新所有 **eventListener** 定义，以使用 **ref** 字段。

注意

从 Pipelines 1.3.x 和早期版本升级到 Pipelines 1.4.0 会中断事件监听程序，因为 **template.name** 字段不可用。在这种情况下，使用 Pipelines 1.4.1 来提供恢复的 **template.name** 字段。

- 对于 **EventListener** 自定义资源/对象，**PodTemplate** 和 **ServiceType** 字段已弃用，并使用 **Resource** 替代。
- 过时的 spec 风格内嵌绑定已被删除。
- **spec** 字段已从 **triggerSpecBinding** 中删除。
- 事件 ID 已从包括五个字符的随机字符串改为 UUID。

4.1.9.4. 已知问题

- 在 **Developer** 视角中，管道指标和触发器功能仅适用于 OpenShift Container Platform 4.7.6 或更高版本。
- 在 IBM Power Systems、IBM Z 和 LinuxONE 中，不支持 **tkn hub** 命令。
- 当您在 IBM Power Systems（ppc64le）、IBM Z 和 LinuxONE（s390x）集群上运行 Maven 和 Jib Maven 集群任务时，将 **MAVEN_IMAGE** 参数值设置为 **maven:3.6.3-adoptopenjdk-11**。
- 如果您在触发器绑定中有以下配置，触发器会因为不正确处理 JSON 格式抛出错误：

```
params:
 - name: github_json
   value: $(body)
```

  要解决这个问题：

  - 如果您使用触发器 v0.11.0 及更高版本，请使用 **marshalJSON** CEL 函数，该函数使用 JSON 对象或数组，并将该对象或数组的 JSON 编码作为字符串返回。
  - 如果使用旧的触发器版本，请在触发器模板中添加以下注解：

```
annotations:
  triggers.tekton.dev/old-escape-quotes: "true"
```

- 当从 Pipelines 1.3.x 升级到 1.4.x 时，您必须重新创建路由。

4.1.9.5. 修复的问题

- 在以前的版本中，**tekton.dev/task** 标签已从集群任务运行中删除，并且引进了 **tekton.dev/clusterTask** 标签。此更改导致的问题可以通过修复 **clustertask describe** 和 **delete** 命令来解决。另外，也修改了任务的 **lastrun** 功能，从而解决应用到在旧版管道中运行任务中的 **tekton.dev/task** 标签的问题。
- 当进行交互式 **tkn pipeline start pipelinename** 时，会以互动方式创建一个 **PipelineResource**。如果资源状态不是 **nil**，**tkn p start** 命令会输出资源状态。
- 在以前的版本中，**tekton.dev/task=name** 标签已从集群任务创建的任务中删除。在这个版本中，使用 **--last** 标志修改 **tkn clustertask start** 命令，以检查所创建的任务运行中的 **tekton.dev/task=name** 标签。
- 当任务使用内联任务规格时，在运行 **tkn pipeline describe** 命令时对应的任务运行会被嵌入到管道中，任务名称返回为内嵌。
- 修复了 **tkn version** 命令，显示已安装的 Tekton CLI 工具的版本，无需配置的 **kubeConfiguration** 命名空间或对集群的访问。
- 如果使用意外的参数或者使用多个参数，则 **tkn completion** 命令会出错。
- 在以前的版本中，当管道转换为 **v1alpha1** 版本并恢复到 **v1beta1** 版本时，带有嵌套在一个管道规格中的 **finally** 的任务将会丢失那些 **finally** 任务。修复了转换过程中发生的这个错误，以避免潜在的数据丢失。现在，带有嵌套在管道规格中的 **finally** 任务的管道运行会被序列化并存储在 alpha 版本中，它们只会在以后进行反序列化。
- 在以前的版本中，当服务帐户中的 **secrets** 字段被设置为 **{}** 时，pod 生成中会出现一个错误。任务运行失败，显示 **CouldntGetTask**，因为带有空 secret 名称的 GET 请求返回了一个错误，表示资源名称可能不是空的。这个问题已通过避免 **kubeclient** GET 请求中的空 secret 名称来解

决。

- 现在，可以请求带有 **v1beta1** API 版本的管道和 **v1alpha1** 版本，而不会丢失 **finall** 任务。应用返回的 **v1alpha1** 版本将以 **v1beta1** 来保存资源，**finally** 部分恢复到其原始状态。

- 在以前的版本中，控制器中的一个未设置的 **selfLink** 字段在 Kubernetes v1.20 集群中造成错误。作为一个临时修复，在没有自动填充的 **selfLink** 字段的值时，**CloudEvent** source 字段被设置为与当前源 URI 匹配的值。

- 在以前的版本中，带有点（如 **gcr.io** ）的 secret 名称会导致任务运行创建失败。这是因为内部使用的 secret 名称作为卷挂载名称的一部分。卷挂载名称遵循 RFC1123 DNS 标签标准，它不允许使用点作为名称的一部分。这个问题已通过将点替换为横线来解决。

- 现在，上下文变量会在 **finally** 任务中进行验证。

- 在以前的版本中，当任务运行协调器传递了一个任务运行，且没有之前的状态更新，其中包含它创建的 pod 的名称时，任务运行协调程序会列出与任务运行关联的 pod。任务运行协调程序使用任务运行标签（被传播到 pod）来查找 pod。在任务运行期间更改这些标签，会导致代码找不到现有的 pod。因此，会创建重复的 pod。这个问题已通过在查找 pod 时将任务运行协调器更改为只使用 **tekton.dev/taskRun** Tekton 控制的标签来解决。

- 在以前的版本中，当管道接受一个可选的工作区并将其传递给管道任务时，如果未提供工作区，管道运行协调器会停止并出错，即使缺少的工作区绑定是可选工作区的有效状态。这个问题已被解决，确保管道运行的协调器不会无法创建任务运行，即使未提供可选的工作区。

- 排序步骤状态的顺序与步骤容器的顺序匹配。

- 在以前的版本中，当 Pod 遇到 **CreateContainerConfigError** 原因时，任务运行状态被设置为 **unknown**，这意味着任务和管道会运行，直到 pod 超时为止。这个问题已通过将任务运行状态设置为 **false** 来解决这个问题，因此当 pod 遇到 **CreateContainerConfigError** 原因时，任务被设置为 failed。

- 在以前的版本中，管道结果会在管道运行完成后在第一次协调时解析。这可能会导致管道运行的 **Succeeded** 条件被覆盖。因此，最终状态信息丢失，可能会使监视管道运行条件的任何服务混淆。当管道运行进入 **Succeeded** 或 **True** 条件时，这个问题可以通过将管道结果解析移到协调结束时来解决。

- 现在，执行状态变量已被验证。这可避免在验证上下文变量来访问执行状态时验证任务结果。

- 在以前的版本中，包含无效变量的管道结果将添加到管道运行中，并包含变量的字面表达式。因此，很难评估结果是否正确填充。这个问题已通过过滤管道运行结果来解决，该结果引用了失败的任务运行。现在，包含无效变量的管道结果将完全不会被管道运行发送。

- 现在，**tkn eventlistener describe** 命令已被修复，以避免在没有模板的情况下崩溃。它还显示有关触发器引用的详情。

- 由于 **template.name** 不可用，从 Pipelines 1.3.x 及早期版本升级到 Pipelines 1.4.0 会破坏事件监听程序。在 Pipelines 1.4.1 中，**template.name** 已恢复，以避免触发器中的事件监听程序。

- 在 Pipelines 1.4.1 中，**ConsoleQuickStart** 自定义资源已更新，以符合 OpenShift Container Platform 4.7 的功能和行为。

4.1.10. Red Hat OpenShift Pipelines 技术预览 1.3 发行注记

4.1.10.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.3 现在包括在 OpenShift Container Platform 4.7 中。Red Hat OpenShift Pipelines TP 1.3 更新为支持：

- Tekton Pipelines 0.19.0
- Tekton **tkn** CLI 0.15.0
- Tekton Triggers 0.10.2
- 基于 Tekton Catalog 0.19.0 的集群任务
- OpenShift Container Platform 4.7 中的 IBM Power Systems
- OpenShift Container Platform 4.7 上的 IBM Z 和 LinuxONE

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.3 中的新内容。

4.1.10.1.1. Pipelines

- 构建镜像的任务，如 S2I 和 Buildah 任务，现在发出构建的镜像 URL，其中包含镜像 SHA。
- 由于 **Conditions** 自定义资源定义（CRD）已弃用，管道任务中引用自定义资源的条件会被禁止。
- 现在，在 **Task** CRD 中为以下字段添加了变量扩展： **spec.steps[].imagePullPolicy** 和 **spec.sidecar[].imagePullPolicy**。
- 您可以通过将 **disable-creds-init** feature-flag 设置为 **true** 来禁用 Tekton 中的内置凭证机制。
- 现在，当在 **PipelineRun** 配置的 **Status** 字段中的 **Skipped Tasks** 和 **Task Runs** 部分中列出了表达式时，可以解析。
- **git init** 命令现在可以克隆递归子模块。
- 现在，**Task** CR 的作者可以为 **Task** spec 的一个步骤指定超时。
- 现在，您可以将入口点镜像基于 **distroless/static:nonroot** 镜像，赋予将其复制到目的地的模式，而无需依赖基础镜像中存在的 **cp** 命令。
- 现在，您可以使用配置标记 **require-git-ssh-secret-known-hosts** 来禁止在 Git SSH secret 中省略已知主机。当标志值设为 **true** 时，必须在 Git SSH secret 中包含 **known_host** 字段。标志的默认值为 **false**。
- 现在引进了可选工作区的概念。任务或管道可能会声明一个工作区（workspace），并有条件地更改其行为。任务运行或管道运行可能省略了工作区，因此修改任务或管道行为。默认任务运行工作区不会添加到忽略的可选工作区。
- 现在，在 Tekton 中进行凭证初始化会检测一个与非 SSH URL 搭配使用的 SSH 凭证，而 Git pipeline 资源与 Git pipeline 资源相同，并在步骤容器中记录警告。
- 如果 pod 模板指定的关联性被关联性代理覆盖，则任务运行控制器会发出警告事件。
- 任务运行协调程序现在记录了在任务运行完成后发送的云事件的指标。这包括重试。

4.1.10.1.2. Pipelines CLI

- 现在，在以下命令中添加了对 **--no-headers flag** 的支持： **tkn condition list**、**tkn triggerbinding list**、**tkn eventlistener list**、**tkn clustertask list**、**tkn clustertriggerbinding list**。
- 当一起使用时，**--last** 或 **--use** 选项会覆盖 **--prefix-name** 和 **--timeout** 选项。
- 现在，添加了 **tkn eventlistener logs** 命令来查看 **EventListener** 日志。
- **tekton hub** 命令现在被集成到 **tkn** CLI。
- **--nocolour** 选项现在改为 **--no-color**。
- **--all-namespaces** 标志添加到以下命令中： **tkn triggertemplate list**、**tkn condition list**、**tkn triggerbinding list**、**tkn eventlistener list**。

4.1.10.1.3. 触发器

- 现在，您可以在 **EventListener** 模板中指定资源信息。
- 现在，**EventListener** 服务帐户除具有所有触发器资源的 **get** verb 外，还具有 **list** 和 **watch** verb。这可让您使用 **Listers** 从 **EventListener**、**Trigger**、**TriggerBinding**、**TriggerTemplate** 和 **ClusterTriggerBinding** 资源中获取数据。您可以使用此功能创建 **Sink** 对象，而不是指定多个通知器，直接向 API 服务器发出调用。
- 添加了一个新的 **Interceptor** 接口，以支持不可变的输入事件正文。拦截器现在可以在一个新的 **extensions** 字段中添加数据或字段，且无法修改输入正文使其不可变。CEL 拦截器使用这个新的 **Interceptor** 接口。
- 在 **EventListener** 资源中添加了一个 **namespaceSelector** 字段。使用它来指定 **EventListener** 资源可以从中获取用于处理事件的 **Trigger** 对象的命名空间。要使用 **namespaceSelector** 字段，**EventListener** 资源的服务帐户必须具有集群角色。
- 触发器 **EventListener** 资源现在支持到 **eventlistener** pod 的端到端安全连接。
- 在 **TriggerTemplates** 资源中把 **"** 替换为 **\"** 的转义行为现在已被删除。
- 一个支持 Kubernetes 资源的、新的 **resources** 项已作为 **EventListener** spec 的一部分被添加。
- 添加了对 CEL 拦截器的新功能，它支持 ASCII 字符串的大写和小写。
- 您可以使用触发器中的 **name** 和 **value** 字段，或事件监听程序来嵌入 **TriggerBinding** 资源。
- **PodSecurityPolicy** 配置已更新，可在受限环境中运行。它确保容器必须以非 root 运行。另外，使用 Pod 安全策略的基于角色的访问控制也从集群范围移到命名空间范围。这样可确保触发器无法使用与命名空间不相关的其他 Pod 安全策略。
- 现在，添加了对内嵌触发器模板的支持。您可以使用 **name** 字段来指代嵌入的模板，或者在 **spec** 字段中嵌入模板。

4.1.10.2. 已弃用的功能

- 使用 **PipelineResources** CRD 的管道模板现已弃用，并将在以后的发行版本中删除。
- **template.name** 字段已弃用，被 **template.ref** 字段替代，并将在以后的发行版本中删除。
- 使用 **-c** 作为 **--check** 命令的缩写已被删除。另外，全局 **tkn** 标志被添加到 **version** 命令中。

4.1.10.3. 已知问题

- CEL 覆盖在新的顶层 **extensions** 功能中添加字段，而不是修改传入的事件正文。**TriggerBinding** 资源可以使用 **$(extensions.<key>)** 语法访问这个新 **extensions** 功能中的值。更新您的绑定，使用 **$(extensions.<key>)** 语法而不是 **$(body.<overlay-key>)** 语法。

- 把 **"** 替换为 **\"** 的参数转义行为现在已被删除。如果您需要保留旧的转义参数行为，请在 **TriggerTemplate** 规格中添加 **tekton.dev/old-escape-quotes: true"** 注解。

- 您可以使用触发器中的 **name** 和 **value** 字段，或事件监听程序来嵌入 **TriggerBinding** 资源。但是，您无法为单个绑定指定 **name** 和 **ref** 字段。使用 **ref** 字段引用 **TriggerBinding** 资源以及内嵌绑定的 **name** 字段。

- 拦截器无法试图引用 **EventListener** 资源命名空间以外的 **secret**。您必须在 'EventListener' 资源的命名空间中包含 secret。

- 在 Triggers 0.9.0 及之后的版本中，如果基于正文或标头的 **TriggerBinding** 参数在事件有效负载中缺失或格式不正确，则使用默认值而不是显示错误。

- 通过使用 Tekton Pipelines 0.16.x 的 **WhenExpression** 对象创建的任务和管道必须重新应用来修复它们的 JSON 注解。

- 当管道接受一个可选的工作区，并将其提供给某个任务时，如果未提供工作区，管道会运行停止。

- 要在断开连接的环境中使用 Buildah 集群任务，请确保 Dockerfile 使用作为基础镜像的内部镜像流，然后使用与任何 S2I 集群任务相同的方法。

4.1.10.4. 修复的问题

- CEL 拦截器添加的扩展通过在事件正文中添加 **Extensions** 字段传递给 Webhook 拦截器。

- 现在，日志读取器的活动超时可以使用 **LogOptions** 字段进行配置。但是，10 秒的默认超时行为会被保留。

- 当一个任务运行或管道运行完成后，**log** 命令会忽略 **--follow** 标记，它会读取可用的日志而不是实时的日志。

- 对以下 Tekton 资源的引用现在标准化，并在 **tkn** 命令中的所有面向用户的信息中保持一致：**EventListener**、**TriggerBinding**、**ClusterTriggerBinding**、**Condition** 和 **TriggerTemplate**。

- 在以前的版本中，如果您启动了使用 **--use-taskrun <canceled-task-run-name>**、**--use-pipelinerun <canceled-pipeline-run-name>** 或 **--last** 标记的、已被取消的任务运行或管道运行，新的运行将会被取消。这个程序漏洞现已解决。

- 现在，**tkn pr desc** 命令已被改进，以便在管道有条件运行时不会失败。

- 当使用 **--task** 选项删除使用 **tkn tr delete** 运行的任务时，且集群任务具有相同名称的集群任务存在时，集群任务运行的任务也会被删除。作为临时解决方案，使用 **TaskRefKind** 字段过滤运行的任务。

- **tkn triggertemplate describe** 命令在输出中只会显示 **apiVersion** 值的一部分。例如，只显示 **triggers.tekton.dev**，而不是 **triggers.tekton.dev/v1alpha1**。这个程序漏洞现已解决。

- 在某些情况下，webhook 无法获取租期且无法正常工作。这个程序漏洞现已解决。

- 在 v0.16.3 中创建的带有 when 表达式的管道现在可以在 v0.17.1 及之后的版本中运行。升级后，您不需要重新应用之前版本中创建的管道定义，因为现在支持注解中第一个字母的大写和小写。
- 默认情况下，**leader-election-ha** 字段为高可用性启用。当 **disable-ha** 控制器标记设置为 **true** 时，它会禁用高可用性支持。
- 现在，解决了重复云事件的问题。现在，只有在条件改变状态、原因或消息时才发送云事件。
- 当 **PipelineRun** 或 **TaskRun** spec 中没有服务帐户名称时，控制器会使用 **config-defaults** 配置映射中的服务帐户名称。如果在 **config-defaults** 配置映射中也缺少服务帐户名称，控制器现在会在 spec 中将其设置为 **default**。
- 现在，当同一个持久性卷声明用于多个工作区，但具有不同子路径时，支持验证是否与关联性偏好兼容。

4.1.11. Red Hat OpenShift Pipelines 技术预览 1.2 发行注记

4.1.11.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.2 现在包括在 OpenShift Container Platform 4.6 中。Red Hat OpenShift Pipelines TP 1.2 更新为支持：

- Tekton Pipelines 0.16.3
- Tekton **tkn** CLI 0.13.1
- Tekton Triggers 0.8.1
- 基于 Tekton Catalog 0.16 的集群任务
- OpenShift Container Platform 4.6 中的 IBM Power Systems
- OpenShift Container Platform 4.6 上的 IBM Z 和 LinuxONE

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.2 中的新内容。

4.1.11.1.1. Pipelines

- 此 Red Hat OpenShift Pipelines 发行版本添加了对断开连接的安装的支持。

注意

IBM Power Systems、IBM Z 和 LinuxONE 目前不支持在受限环境中安装。

- 现在，您可以使用 **when** 字段而不是 **conditions** 资源，仅在满足特定条件时运行任务。**WhenExpression** 资源的关键组件是 **Input**、**Operator** 和 **Values**。如果所有表达式都评估的结果都为 **True**，则任务运行。如果表达式评估的结果为 **False**，则任务被跳过。
- 现在，如果某个任务运行被取消或超时，则步骤（Step）状态被更新。
- 现在，支持 Git 大文件存储（LFS）来使用 **git-init** 构建基础镜像。
- 现在，当某个任务嵌入到管道中时，您可以使用 **taskSpec** 字段来指定元数据，如标识（label）和注解（annotation）。

- 现在，Pipeline 运行支持云事件。现在，对于云事件管道资源发送的带有 **backoff** 的云事件会进行重试。
- 现在，可以为声明了 **Task**、但没有明确指定 **TaskRun** 资源的工作区（workspace）设置一个默认的 **Workspace** 配置。
- 支持 **PipelineRun** 命名空间和 **TaskRun** 命名空间的命名空间变量插入。
- 现在，添加了对 **TaskRun** 对象的验证，以检查当 **TaskRun** 资源与 Affinity Assistant 关联时，是否使用一个以上的持久性卷声明工作区。如果使用多个持久性卷声明工作区，则任务运行会失败，并且有一个 **TaskRunValidationFailed** 条件。请注意，默认情况下， Affinity Assistant 在 Red Hat OpenShift Pipelines 中被禁用，因此您需要启用 Affinity Assistant 来使用它。

4.1.11.1.2. Pipelines CLI

- **tkn task describe**、**tkn taskrun describe**、**tkn clustertask describe**、**tkn pipeline describe** 和 **tkn pipelinerun describe** 命令现在：
  - 如果存在其中之一，会自动选择 **Task**、**TaskRun**、**ClusterTask**、**Pipeline** 和 **PipelineRun**。
  - 在相应的输出中显示 **Task**、**TaskRun**、**ClusterTask**、**Pipeline** 和 **PipelineRun** 资源的结果。
  - 在相应的输出中显示 **Task**、**TaskRun**、**ClusterTask**、**Pipeline** 和 **PipelineRun** 资源中声明的工作区。
- 现在，您可以使用 **tkn clustertask start** 命令的 **--prefix-name** 选项指定任务运行名称前缀。
- 现在为 **tkn clustertask start** 命令提供了互动模式支持。
- 现在，您可以使用 **TaskRun** 和 **PipelineRun** 对象的本地或远程文件定义指定管道支持的 **PodTemplate** 属性。
- 现在，您可以在 **tkn clustertask start** 命令中使用 **--use-params-defaults** 选项，使用 **ClusterTask** 配置中设置的默认值并创建任务运行。
- 现在，如果有些参数没有指定默认值，**tkn pipeline start** 命令的 **--use-param-defaults** 标志会提示以互动模式提供。

4.1.11.1.3. 触发器

- 添加了一个名为 **parseYAML** 的通用表达语言（CEL）函数，用来将 YAML 字符串解析为一个映射的字符串。
- 在评估表达式和解析 hook 正文以创建评估环境时，改进了解析 CEL 表达式的错误消息，使其更加精细。
- 现在，可以支持 marsing 布尔值和映射，如果它们被用作 CEL 覆盖机制中的表达式值。
- 在 **EventListener** 对象中添加了以下字段：
  - **replicas** 字段通过在 YAML 文件中指定副本数，使事件监听程序能够运行多个 pod。
  - **NodeSelector** 字段使 **EventListener** 对象能够将事件监听器 pod 调度到特定的节点。

- Webhook 拦截器现在可以解析 **EventListener-Request-URL** 标头，从事件监听器处理的原始请求 URL 中提取参数。
- 现在，事件监听器的注解可以被传播到部署、服务和其他 pod。请注意，服务或部署的自定义注解将被覆盖，因此必须在事件监听程序注解中添加它们以便传播它们。
- 现在，当用户将 **spec.replicas** 值指定为 **负数** 或 **零**时，可以正确验证 **EventListener** 规格中的副本。
- 现在，您可以在 **EventListener** spec 中指定 **TriggerCRD** 项，作为一个使用 **TriggerRef** 项的引用来独立创建 **TriggerCRD** 项，然后在**EventListener** spec 中绑定它。
- 现在，提供了对 **TriggerCRD** 对象的验证和默认值。

4.1.11.2. 已弃用的功能

- **$(params)** 参数现已从 **triggertemplate** 资源中删除，由 **$(tt.params)** 替代，以避免 **resourcetemplate** 和 **triggertemplate** 资源参数间的混淆。
- 基于可选的基于 **EventListenerTrigger** 的身份验证级别的 **ServiceAccount** 引用，已从对象引用改为一个 **ServiceAccountName** 字符串。这样可确保 **ServiceAccount** 引用与 **EventListenerTrigger** 对象位于同一个命名空间中。
- **Conditions** 自定义资源定义（CRD）现已弃用，已使用 **WhenExpressions** CRD 替代。
- **PipelineRun.Spec.ServiceAccountNames** 对象已启用，被 **PipelineRun.Spec.TaskRunSpec[].ServiceAccountName** 对象替代。

4.1.11.3. 已知问题

- 此 Red Hat OpenShift Pipelines 发行版本添加了对断开连接的安装的支持。但是，集群任务使用的一些镜像必须进行镜像（mirror）才能在断开连接的集群中工作。
- 在卸载 Red Hat OpenShift Pipelines Operator 后，**openshift** 命名空间中的管道不会被删除。使用 **oc delete pipelines -n openshift --all** 命令删除管道。
- 卸载 Red Hat OpenShift Pipelines Operator 不会删除事件监听程序。
  作为临时解决方案，删除 **EventListener** 和 **Pod** CRD:

  1. 使用 **foregroundDeletion** 终结器编辑 **EventListener** 对象：

     ```
     $ oc patch el/<eventlistener_name> -p '{"metadata":{"finalizers":["foregroundDeletion"]}}'
     --type=merge
     ```

     例如：

     ```
     $ oc patch el/github-listener-interceptor -p '{"metadata":{"finalizers":
     ["foregroundDeletion"]}}' --type=merge
     ```

  2. 删除 **EventListener** CRD:

     ```
     $ oc patch crd/eventlisteners.triggers.tekton.dev -p '{"metadata":{"finalizers":[]}}' --
     type=merge
     ```

- 当您运行多架构容器镜像任务时，如果在 IBM Power Systems(ppc64le)或 IBM Z(s390x)集群上没有命令规格，则 **TaskRun** 资源会失败，并显示以下错误：

```
Error executing command: fork/exec /bin/bash: exec format error
```

作为临时解决方案，使用特定架构的容器镜像或指定 sha256 摘要指向正确的架构。要获得 sha256 摘要，请输入：

```
$ skopeo inspect --raw <image_name>| jq '.manifests[] | select(.platform.architecture == "
<architecture>") | .digest'
```

4.1.11.4. 修复的问题

- 现在，添加了一个简单的语法验证用于检查 CEL 过滤器、Webhook 验证器中的覆盖以及拦截器中的表达式。
- 触发器不再覆盖底层部署和服务对象上的注解。
- 在以前的版本中，事件监听器将停止接受事件。**EventListener** sink 增加了一个 120 秒的空闲超时来解决这个问题。
- 在以前的版本中，取消一个带有 **Failed(Canceled)** 状态的管道运行会给出一个成功信息。这个问题已被解决，现在在这种情况下会显示错误。
- **tkn eventlistener list** 命令现在提供列出的事件监听器的状态，从而使您可以轻松地识别可用的事件。
- 现在，当没有安装触发器或没有找到资源时，**triggers list** 和 **triggers describe** 命令会显示一致的错误信息。
- 在以前的版本中，在云事件交付过程中会产生大量闲置连接。**DisableKeepAlives: true** 参数添加到 **cloudeventclient** 配置来修复这个问题。因此，会为每个云事件设置一个新的连接。
- 在以前的版本中，**creds-init** 代码也会向磁盘写入空文件，即使未提供给定类型的凭证。在这个版本中，**creds-init** 代码只为从正确注解的 secret 中挂载的凭证写入文件。

4.1.12. Red Hat OpenShift Pipelines 技术预览 1.1 发行注记

4.1.12.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.1 现在包括在 OpenShift Container Platform 4.5 中。Red Hat OpenShift Pipelines TP 1.1 更新为支持：

- Tekton Pipelines 0.14.3
- Tekton **tkn** CLI 0.11.0
- Tekton Triggers 0.6.1
- 基于 Tekton Catalog 0.14 的集群任务

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.1 中的新内容。

4.1.12.1.1. Pipelines

- 现在可以使用工作区而不是管道资源。建议您在 OpenShift Pipelines 中使用 Workspaces 而不是 PipelineResources，因为 PipelineResources 很难调试，范围有限，且不容易重复使用。如需有关 Workspaces 的更多信息，请参阅了解 OpenShift Pipelines。

- 添加了对卷声明模板的工作空间支持：

  - 管道运行和任务运行的卷声明模板现在可以添加为工作区的卷源。然后，tkton-controller 使用模板创建一个持久性卷声明（PVC），该模板被视为管道中运行的所有任务的 PVC。因此，您不需要在每次绑定多个任务的工作空间时都定义 PVC 配置。

  - 当卷声明模板用作卷源时，支持使用变量替换来查找 PVC 名称。

- 支持改进的审核：

  - **PipelineRun.Status** 字段现在包含管道中运行的每个任务的状态，以及用于实例化用于监控管道运行进度的管道规格。

  - Pipeline 结果已添加到 pipeline 规格和 **PipelineRun** 状态中。

  - **TaskRun.Status** 字段现在包含用于实例化 **TaskRun** 资源的具体任务规格。

- 支持在条件中应用默认参数。

- 现在，通过引用集群任务创建的任务运行会添加 **tekton.dev/clusterTask** 标签，而不是 **tekton.dev/task** 标签。

- kube config writer 现在在资源结构中添加了 **ClientKeyData** 和 **ClientCertificateData** 配置，以便使用 kubeconfig-creator 任务替换 pipeline 资源类型集群。

- 现在，**feature-flags** 和 **config-defaults** 配置映射的名称可以自定义。

- 现在，在任务运行使用的 pod 模板中支持主机网络。

- 现在，可以使用 Affinity Assistant 支持任务运行中共享工作空间卷的节点关联性。默认情况下，这在 OpenShift Pipelines 上被禁用。

- Pod 模板已更新，使用 **imagePullSecrets** 指定在启动一个 pod 时，容器运行时用来拉取容器镜像的 secret。

- 如果控制器无法更新任务运行，则支持从任务运行控制器发出警告事件。

- 在所有资源中添加了标准或者推荐的 k8s 标签，以标识属于应用程序或组件的资源。

- 现在，**Entrypoint** 进程被通知有信号，然后这些信号会使用一个 **Entrypoint** 进程的专用 PID 组来传播这些信号。

- pod 模板现在可以在运行时使用任务运行 specs 在任务级别设置。

- 支持放出 Kubernetes 事件：

  - 控制器现在会为其他任务运行生命周期事件发出事件 - **taskrun started** 和 **taskrun running**。

  - 频道运行控制器现在会在管道每次启动时放出一个事件。

- 除了默认的 Kubernetes 事件外，现在还提供对任务运行的支持。可将控制器配置为发送任何任务运行事件（如创建、启动和失败）作为云事件。

- 支持使用 **$context.<task|taskRun|pipelineRun>.name** 变量来引用管道运行和任务运行时的适当名称。
- 现在提供了管道运行参数的验证，以确保管道运行提供了管道所需的所有参数。这也允许管道运行在所需参数之外提供额外的参数。
- 现在，您可以使用管道 YAML 文件中的 **finally** 字段指定管道中的任务，这些任务会在管道退出前始终执行。
- **git-clone** 集群任务现在可用。

4.1.12.1.2. Pipelines CLI

- **tkn dlistener describe** 命令现在可以支持内嵌触发器绑定。
- 支持在使用不正确的子命令时推荐子命令并给出建议。
- 现在，如果管道中只有一个任务存在，**tkn task describe** 命令会自动选择该任务。
- 现在您可以使用默认参数值启动任务，方法是在 **tkn task start** 命令中指定 **--use-param-defaults** 标记。
- 现在，您可以使用 **tkn pipeline start** 或 **tkn task start** 命令的 **--workspace** 选项为管道运行或任务指定卷声明模板。
- **tkn pipelinerun logs** 命令现在会显示 **finally** 部分中列出的最终任务的日志。
- 现在，为 **tkn task start** 命令提供了互动模式支持，并为以下 **tkn** 资源提供 **describe** 子命令：**pipeline**, **PipelineRun**、**task**、**taskrun**、**clustertask** 和 **pipelineresource**。
- **tkn version** 命令现在显示集群中安装的触发器版本。
- **tkn pipeline describe** 命令现在显示为频道中使用的任务指定的参数值和超时。
- 添加了对 **tkn pipelinerun describe** 和 **tkn taskrun describe** 命令的 **--last** 选项的支持，以分别描述最新的频道运行或任务运行。
- **tkn pipeline describe** 命令现在显示管道中适用于任务的条件。
- 现在，您可以在 **tkn resource list** 命令中使用 **--no-headers** 和 **--all-namespaces** 标记。

4.1.12.1.3. 触发器

- 现在以下通用表达式语言（CEL）功能可用：
  - **parseURL** 用来解析和提取一个 URL 的部分内容
  - **parseJSON** 用来解析嵌入在 **deployment** webhook 中的 **payload** 字段中的字符串中的 JSON 值类型
- 添加了来自 Bitbucket 的 webhook 的新拦截器。
- 现在，在使用 **kubectl get** 列出时，事件监听器会显示 **Address URL** 和 **Available status** 作为额外的项。
- 触发器模板参数现在使用 **$(tt.params.<paramName>)** 语法而不是 **$(params.<paramName>)** 来减少触发器模板和资源模板参数之间的混淆。

- 现在，您可以在 **EventListener** CRD 中添加 **容限**，以确保事件监听程序使用相同的配置，即使所有节点都因为安全或管理问题而产生污点也是如此。
- 现在，您可以在 **URL/live** 中为事件监听器添加就绪探测（Readiness Probe）。
- 现在，添加了对在事件监听器触发器中嵌入 **TriggerBinding** 规格的支持。
- 触发器资源现在附带推荐的 **app.kubernetes.io** 标签注解。

4.1.12.2. 已弃用的功能

本发行版本中已弃用了以下内容：

- 所有集群范围命令（包括 **clustertask** 和 **clustertriggerbinding** 命令）的 **--namespace** 或 **-n** 标志都已弃用。它将在以后的发行版本中被删除。
- 事件监听器中的 **triggers.bindings** 中的 **name** 字段已弃用。现在使用 **ref** 字段替代，并将在以后的发行版本中删除。
- 使用 **$(params)** 的触发器模板中的变量插值已经被弃用，现在使用 **$(tt.params)** 来减少与管道变量插入语法的混乱。在以后的发行版本中会删除 **$(params.<paramName>)** 语法。
- 在集群任务中弃用了 **tekton.dev/task** 标签。
- **TaskRun.Status.ResourceResults.ResourceRef** 字段已弃用，并将被删除。
- **tkn pipeline create**、**tkn task create** 和 **tkn resource create -f** 子命令已被删除。
- 从 **tkn** 命令中删除了命名空间验证。
- **tkn ct start** 命令中的默认超时时间（**1h**）以及 **-t** 标志已被删除。
- **s2i** 集群任务已弃用。

4.1.12.3. 已知问题

- 条件（Conditions）不支持工作区。
- **tkn clustertask start** 命令不支持 **--workspace** 选项和互动模式。
- 支持 **$(params.<paramName>)** 语法的向后兼容性会强制您使用带有特定管道参数的触发器模板，因为触发器 s Webhook 无法将触发器参数与管道参数区分开。
- 当针对 **tekton_taskrun_count** 和 **tekton_taskrun_duration_seconds_count** 运行一个 promQL 查询时，Pipeline metrics 会报告不正确的值。
- 当为一个工作区指定了一个不存在的 PVC 名称时，管道运行和任务运行会维持在 **Running** 和 **Running(Pending)** 的状态。

4.1.12.4. 修复的问题

- 在以前的版本中，如果任务和集群任务的名称是相同的，则 **tkn task delete <name> --trs** 命令会同时删除 Task 和 ClusterTask。在这个版本中，该命令只删除任务 **<name>** 创建的任务运行。

- 以前，**tkn pr delete -p <name> --keep 2** 命令会在使用 **--keep** 是忽略 **-p** 标志，并将删除除最后两个以外的所有管道运行。在这个版本中，命令只删除由管道 **<name>** 创建的管道运行，但最后两个除外。
- **tkn triggertemplate describe** 输出现在以表格式而不是 YAML 格式显示资源模板。
- 在以前的版本中，当一个新用户添加到容器时，**buildah** 集群任务会失败。在这个版本中，这个问题已被解决。

4.1.13. Red Hat OpenShift Pipelines 技术预览 1.0 发行注记

4.1.13.1. 新功能

Red Hat OpenShift Pipelines 技术预览（TP）1.0 现在包括在 OpenShift Container Platform 4.4 中。Red Hat OpenShift Pipelines TP 1.0 更新为支持：

- Tekton Pipelines 0.11.3
- Tekton **tkn** CLI 0.9.0
- Tekton Triggers 0.4.0
- 基于 Tekton Catalog 0.11 的集群任务

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift Pipelines 1.0 中的新内容。

4.1.13.1.1. Pipelines

- 支持 v1beta1 API 版本。
- 支持改进的限制范围。在以前的版本中，限制范围仅为任务运行和管道运行指定。现在不需要显式指定限制范围。使用命名空间中的最小限制范围。
- 支持使用任务结果和任务参数在任务间共享数据。
- 现在，管道可以被配置为不覆盖 **HOME** 环境变量和步骤的工作目录。
- 与任务步骤类似，**sidecar** 现在支持脚本模式。
- 现在，您可以在任务运行 **podTemplate** 资源中指定不同的调度程序名称。
- 支持使用 Star Array Notation 替换变量。
- Tekton 控制器现在可以配置为监控单个命名空间。
- 现在，在管道、任务、集群任务、资源和条件规格中添加了一个新的 description 字段。
- 在 Git pipeline 资源中添加代理参数。

4.1.13.1.2. Pipelines CLI

- 现在为以下 **tkn** 资源添加了 **describe** 子命令： **EventListener**、**Condition**、**triggerTemplate**、**ClusterTask** 和 **TriggerSBinding**。

- 在以下资源中添加 **v1beta1** 支持以及 **v1alpha1** 的向后兼容性： **ClusterTask**、**Task**、**Pipeline**、**PipelineRun** 和 **TaskRun**。
- 以下命令现在可以使用 **--all-namespaces** 标志选项列出所有命名空间的输出结果： **tkn task list**、**tkn pipeline list**、**tkn taskrun list** 和 **tkn pipelinerun list**
  这些命令的输出也可以通过 **--no-headers** 选项在没有标头的情况下显示信息。
- 现在您可以使用默认参数值启动管道，方法是在 **tkn pipelines start** 命令中指定 **--use-param-defaults** 标记。
- 现在，在 **tkn pipeline start** 和 **tkn task start** 命令中增加了对工作区的支持。
- 现在增加了一个新命令 **clustertriggerbinding**，它带有以下子命令：**describe**、**delete** 和 **list**。
- 现在，您可以使用本地或远程 **yaml** 文件直接启动管道运行。
- **describe** 子命令现在显示一个改进的详细输出。现在，除了新的项，如 **description**、**timeout**、**param description** 和 **sidecar status**，命令输出还提供了关于一个特定 **tkn** 资源的更详细的信息。
- 现在，如果命名空间中只有一个任务，**tkn task log** 命令会直接显示日志。

4.1.13.1.3. 触发器

- 现在触发器可以同时创建 **v1alpha1** 和 **v1beta1** 管道资源。
- 支持新的通用表达式语言(CEL)拦截器功能 - **compareSecret**。此功能安全地将字符串与 CEL 表达式中的 secret 进行比较。
- 支持在事件监听器触发器级别进行身份验证和授权。

4.1.13.2. 已弃用的功能

本发行版本中已弃用了以下内容：

- **Steps** 规格中的环境变量 **$HOME**，变量 **workingDir** 已被弃用，并可能在以后的发行版本中有所变化。目前，在 **Step** 容器中，**HOME** 和 **workingDir** 变量会分别被 **/tekton/home** 和 **/workspace** 变量覆盖。
  在以后的发行版本中，这两个字段将不会被修改，它将被设置为容器镜像和 **Task** YAML 中定义的值。在本发行版本中，使用 **disable-home-env-overwrite** 和 **disable-working-directory-overwrite** 标记来禁用覆盖 **HOME** 和 **workingDir** 变量。
- 以下命令已弃用，并可能在以后的发行版本中删除： **tkn pipeline create**、**tkn task create**。
- 在 **tkn resource create** 命令中使用 **-f** 标志现已弃用。以后的发行版本中可能会删除它。
- **tkn clustertask create** 命令中的 **-t** 标记和 **--timeout** 标记（使用秒格式）现已被弃用。现在只支持持续超时格式，例如 **1h30s**。这些已弃用的标记可能会在以后的版本中删除。

4.1.13.3. 已知问题

- 如果您要从 Red Hat OpenShift Pipelines 的旧版本升级，则必须删除您现有的部署，然后再升级到 Red Hat OpenShift Pipelines 版本 1.0。要删除现有的部署，您必须首先删除自定义资源，然后卸载 Red Hat OpenShift Pipelines Operator。如需了解更多详细信息，请参阅卸载 Red Hat OpenShift Pipelines 部分。

- 提交相同的 **v1alpha1** 任务多次会导致错误。在重新提交一个 **v1alpha1** 任务时，使用 **oc replace** 命令而不是 **oc apply**。

- 当一个新用户添加到容器时，**buildah** 集群任务无法正常工作。
当安装 Operator 时，**buildah** 集群任务的 **--storage-driver** 标志没有指定，因此它会被设置为默认值。在某些情况下，这会导致存储驱动程序设置不正确。当添加一个新用户时，错误的 storage-driver 会造成 **buildah** 集群任务失败并带有以下错误：

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

作为临时解决方案，在 **buildah-task.yaml** 文件中手工把 **--storage-driver** 标识的值设置为 **overlay**：

1. 以 **cluster-admin** 身份登录到集群:

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. 使用 **oc edit** 命令编辑 **buildah** 集群任务：

```
$ oc edit clustertask buildah
```

buildah clustertask YAML 文件的最新版本会在由 **EDITOR** 环境变量指定的编辑器中打开。

3. 在 **Steps** 字段中找到以下 **command** 字段：

```
 command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-
verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t',
'$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

4. 使用以下内容替换 **command** 字段：

```
 command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--
tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t',
'$(params.IMAGE)', '$(params.CONTEXT)']
```

5. 保存文件并退出。

另外，您还可以直接在 web 控制台中直接修改 **buildah** 集群任务 YAML 文件：进入 **Pipelines → Cluster Tasks → buildah**。从 **Actions** 菜单中选择 **Edit Cluster Task**，如前所示替换 **command** 项。

4.1.13.4. 修复的问题

- 在以前的版本中，即使镜像构建已在进行中，**DeploymentConfig** 任务也会触发新的部署构建。这会导致管道部署失败。在这个版本中，**deploy task** 命令被 **oc rollout status** 命令替代，它会等待正在进行中的部署完成。

- 现在在管道模板中添加了对 **APP_NAME** 参数的支持。

- 在以前的版本中，Java S2I 的管道模板无法在 registry 中查找镜像。在这个版本中，使用现有镜像管道资源而不是用户提供的 **IMAGE_NAME** 参数来查找镜像。

- 所有 OpenShift Pipelines 镜像现在都基于 Red Hat Universal Base Images（UBI）。

- 在以前的版本中，当管道在 **tekton-pipelines** 以外的命名空间中安装时，**tkn version** 命令会将管道版本显示为 **unknown**。在这个版本中，**tkn version** 命令会在任意命名空间中显示正确的管道版本。
- **tkn version** 命令不再支持 **-c** 标志。
- 非管理员用户现在可以列出集群触发器绑定。
- 现在为 CEL 拦截器修复了事件监听程序 **CompareSecret** 功能。
- 现在，当任务和集群任务的名称相同时，任务和集群任务的 **list**、**describe** 和 **start** 子命令可以正确地显示输出。
- 在以前的版本中，OpenShift Pipelines Operator 修改了特权安全性上下文约束 (SCC)，这会在集群升级过程中造成错误。这个错误现已解决。
- 在 **tekton-pipelines** 命名空间中，现在将所有任务运行和管道运行的超时设置为使用配置映射的 **default-timeout-minutes** 字段。
- 在以前的版本中，Web 控制台中的管道部分没有为非管理员用户显示。这个问题现已解决。

4.2. 了解 OPENSHIFT PIPELINES

Red Hat OpenShift Pipelines 是一个基于 Kubernetes 资源的云原生的持续集成和持续交付（continuous integration and continuous delivery，简称 CI/CD）的解决方案。它通过提取底层实现的详情，使用 Tekton 构建块进行跨多个平台的自动部署。Tekton 引入了多个标准自定义资源定义 (CRD)，用于定义可跨 Kubernetes 分布的 CI/CD 管道。

4.2.1. 主要特性

- Red Hat OpenShift Pipelines 是一个无服务器的 CI/CD 系统，它在独立的容器中运行 Pipelines，以及所有需要的依赖组件。
- Red Hat OpenShift Pipelines 是为开发基于微服务架构的非中心化团队设计的。
- Red Hat OpenShift Pipelines 使用标准 CI/CD 管道（pipeline）定义，这些定义可轻松扩展并与现有 Kubernetes 工具集成，可让您按需扩展。
- 您可以通过 Red Hat OpenShift Pipelines 使用 Kubernetes 工具（如 Source-to-Image (S2I)、Buildah、Buildpacks 和 Kaniko）构建镜像，这些工具可移植到任何 Kubernetes 平台。
- 您可以使用 OpenShift Container Platform Web 控制台 **Developer** 视角来创建 Tekton 资源，查看管道运行的日志，并管理 OpenShift Container Platform 命名空间中的管道。

4.2.2. OpenShift Pipeline 概念

本指南提供了对管道（pipeline）概念的详细论述。

4.2.2.1. 任务

Task（任务） 是管道的构建块，它由按顺序执行的步骤组成。它基本上是一个输入和输出的功能。一个任务可以单独运行，也可以作为管道的一部分运行。任务（Task）可以重复使用，并可用于多个 Pipelines。

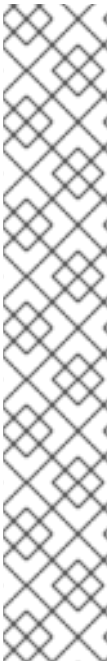
*Step（步骤）*是由任务顺序执行并实现特定目标（如构建镜像）的一系列命令。每个任务都作为 pod 运行，每个步骤都作为该 pod 中的容器运行。由于步骤在同一个 pod 中运行，所以它们可以访问同一卷来缓存文件、配置映射和 secret。

以下示例显示了 **apply-manifests** 任务。

```
apiVersion: tekton.dev/v1beta1 1
kind: Task 2
metadata:
  name: apply-manifests 3
spec: 4
  workspaces:
  - name: source
  params:
    - name: manifest_dir
      description: The directory in source that contains yaml manifests
      type: string
      default: "k8s"
  steps:
    - name: apply
      image: image-registry.openshift-image-registry.svc:5000/openshift/cli:latest
      workingDir: /workspace/source
      command: ["/bin/bash", "-c"]
      args:
        - |-
          echo Applying manifests in $(params.manifest_dir) directory
          oc apply -f $(params.manifest_dir)
          echo -----------------------------------
```

1. Task API 版本 **v1beta1**。
2. Kubernetes 对象的类型，**任务**。
3. 此任务的唯一名称。
4. 列出任务中的参数和步骤，以及任务使用的工作区（workspace）。

此任务启动 pod，并在该 pod 中使用指定镜像运行一个容器，以运行指定的命令。



注意

从 Pipelines 1.6 开始，步骤 YAML 文件中的以下默认值会被删除：

- **HOME** 环境变量不会被默认为 **/tekton/home** 目录
- **workingDir** 字段不会默认为 **/workspace** 目录

相反，步骤的容器定义 **HOME** 环境变量和 **workingDir** 字段。但是，您可以通过在 YAML 文件中为步骤指定自定义值来覆盖默认值。

作为临时方法，为保持与旧 Pipelines 版本向后兼容性，您可以在 **TektonConfig** 自定义资源定义中将以下字段设置为 **false** ：

```
spec:
  pipeline:
    disable-working-directory-overwrite: false
    disable-home-env-overwrite: false
```

4.2.2.2. When 表达式

When 表达式通过设置管道中执行任务的条件来执行任务。它们包含一个组件列表，允许仅在满足特定条件时执行任务。当使用管道 YAML 文件中的 **finally** 字段指定的最终任务集合中也支持表达式。

表达式的主要组件如下：

- **input**：指定静态输入或变量，如参数、任务结果和执行状态。您必须输入有效的输入。如果没有输入有效输入，则其值默认为空字符串。
- **operator**：指定一个输入与一组 **values** 的关系。输入 **in** 或 **notin** 作为 operator 的值。
- **values**：指定字符串值的数组。输入由静态值或变量组成的非空数组，如参数、结果和工作空间的绑定状态。

在任务运行前评估表达式时声明的。如果 when 表达式的值为 **True**，则任务将运行。如果 when 表达式的值为 **False**，则跳过任务。

您可以在各种用例中使用 when 表达式。例如，是否：

- 上一任务的结果如预期所示。
- 之前的提交中更改了 Git 存储库中的文件。
- 镜像是否存在于 registry 中。
- 有可选的工作区可用。

以下示例显示了管道运行的 when 表达式。只有在满足以下条件时，管道运行才会执行 **create-file** 任务：**path** 参数为**README.md**，只有来自 **check-file** 的任务的 **exists** 结果为 **yes** 时才执行 **echo-file-exists** 任务。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun (1)
metadata:
  generateName: guarded-pr-
spec:
```

```
serviceAccountName: 'pipeline'
pipelineSpec:
  params:
    - name: path
      type: string
      description: The path of the file to be created
  workspaces:
    - name: source
      description: |
        This workspace is shared among all the pipeline tasks to read/write common resources
  tasks:
    - name: create-file 2
      when:
        - input: "$(params.path)"
          operator: in
          values: ["README.md"]
      workspaces:
        - name: source
          workspace: source
      taskSpec:
        workspaces:
          - name: source
            description: The workspace to create the readme file in
        steps:
          - name: write-new-stuff
            image: ubuntu
            script: 'touch $(workspaces.source.path)/README.md'
    - name: check-file
      params:
        - name: path
          value: "$(params.path)"
      workspaces:
        - name: source
          workspace: source
      runAfter:
        - create-file
      taskSpec:
        params:
          - name: path
        workspaces:
          - name: source
            description: The workspace to check for the file
        results:
          - name: exists
            description: indicates whether the file exists or is missing
        steps:
          - name: check-file
            image: alpine
            script: |
              if test -f $(workspaces.source.path)/$(params.path); then
                printf yes | tee /tekton/results/exists
              else
                printf no | tee /tekton/results/exists
              fi
    - name: echo-file-exists
      when: 3
```

```
        - input: "$(tasks.check-file.results.exists)"
          operator: in
          values: ["yes"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: 'echo file exists'
...
    - name: task-should-be-skipped-1
      when: 4
        - input: "$(params.path)"
          operator: notin
          values: ["README.md"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: exit 1
...
  finally:
    - name: finally-task-should-be-executed
      when: 5
        - input: "$(tasks.echo-file-exists.status)"
          operator: in
          values: ["Succeeded"]
        - input: "$(tasks.status)"
          operator: in
          values: ["Succeeded"]
        - input: "$(tasks.check-file.results.exists)"
          operator: in
          values: ["yes"]
        - input: "$(params.path)"
          operator: in
          values: ["README.md"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: 'echo finally done'
  params:
    - name: path
      value: README.md
  workspaces:
    - name: source
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 16Mi
```

1 指定 Kubernetes 对象的类型。在本例中，**PipelineRun**。

2 Pipeline 中使用的任务 **create-file**。

3 **when** 表达式指定，只有来自 **check-file** 的 **exists** 结果为 **yes** 时才执行 **echo-file-exists** 任务。

4 **when** 表达式指定，只有 **path** 参数是 **README.md** 时跳过 **task-should-be-skipped-1** 任务。

5 **when** 表达式指定，只有 **echo-file-exists** 任务的执行状态以及任务状态为 **Succeeded**，来自 **check-file** 任务的 **exists** 结果为 **yes**，**path** 参数是 **README.md** 时，才执行 **finally-task-should-be-executed** 任务。

OpenShift Container Platform Web 控制台的 **Pipeline Run details** 页面显示任务和 when 表达式的状态，如下所示：

- 所有条件都满足：任务和 when 表达式符号（以钻石形表示）为绿色。
- 有任何一个条件不符合：任务被跳过。跳过的任务和 when 表达式符号为灰色。
- 未满足任何条件：任务被跳过。跳过的任务和 when 表达式符号为灰色。
- 任务运行失败：失败的任务和 when 表达式符号为红色。

4.2.2.3. 最后的任务

finally 任务是使用管道 YAML 文件中的 **finally** 字段指定的最终任务集合。**finally** 任务始终执行管道中的任务，无论管道运行是否成功执行。**finally** 任务以并行方式执行，在所有管道任务运行后，相应的频道存在前。

您可以配置一个 **finally** 任务，以使用同一管道中任何任务的结果。这个方法不会更改运行此最终任务的顺序。它在所有非最终任务执行后与其他最终任务并行执行。

以下示例显示了 **clone-cleanup-workspace** 管道的代码片段。此代码将存储库克隆到共享的工作区，并清理工作区。执行管道任务后，管道 YAML 文件的 **finally** 中指定的 **cleanup** 任务会清理工作区。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: clone-cleanup-workspace 1
spec:
  workspaces:
    - name: git-source 2
  tasks:
    - name: clone-app-repo 3
      taskRef:
        name: git-clone-from-catalog
      params:
        - name: url
          value: https://github.com/tektoncd/community.git
        - name: subdirectory
          value: application
      workspaces:
        - name: output
          workspace: git-source
  finally:
    - name: cleanup 4
      taskRef: 5
        name: cleanup-workspace
      workspaces: 6
```

```
      - name: source
        workspace: git-source
    - name: check-git-commit
      params: 7
        - name: commit
          value: $(tasks.clone-app-repo.results.commit)
      taskSpec: 8
        params:
          - name: commit
        steps:
          - name: check-commit-initialized
            image: alpine
            script: |
              if [[ ! $(params.commit) ]]; then
                exit 1
              fi
```

1. Pipeline 的唯一名称。
2. 克隆 git 存储库的共享工作区。
3. 将应用存储库克隆到共享工作区的任务。
4. 清理共享工作区的任务。
5. 对 TaskRun 中要执行的任务的引用。
6. Pipeline 中的任务在运行时需要的共享存储卷来接收输入或提供输出。
7. 任务所需的参数列表。如果参数没有隐式默认值，您必须明确设置其值。
8. 嵌入式任务定义.

4.2.2.4. TaskRun

TaskRun 使用集群上的特定输入、输出和执行参数来实例化一个任务用来执行它。它可自行调用，或作为管道中每个任务的 PipelineRun 的一部分。

任务由执行容器镜像的一个或多个步骤组成，每个容器镜像执行特定的构建工作。TaskRun 以指定顺序在任务中执行步骤，直到所有步骤都成功执行或发生失败为止。PipelineRun 由 Pipeline 中每个任务的 PipelineRun 自动创建。

以下示例显示了一个带有相关输入参数运行 **apply-manifests** 任务的 TaskRun:

```
apiVersion: tekton.dev/v1beta1 1
kind: TaskRun 2
metadata:
  name: apply-manifests-taskrun 3
spec: 4
  serviceAccountName: pipeline
  taskRef: 5
    kind: Task
    name: apply-manifests
  workspaces: 6
```

```
  - name: source
    persistentVolumeClaim:
      claimName: source-pvc
```

1. TaskRun API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TaskRun**。
3. 用于标识此 TaskRun 的唯一名称。
4. TaskRun 的定义。对于这个 TaskRun，指定了任务和所需的工作区。
5. 用于此 TaskRun 的任务引用的名称。此 TaskRun 会执行 **apply-manifests** 任务。
6. TaskRun 使用的工作空间。

4.2.2.5. Pipelines

Pipeline 一组 **Task（任务）**资源，它们按特定顺序执行。执行它们是为了构建复杂的工作流，以自动化应用程序的构建、部署和交付。您可以使用包含一个或多个任务的管道为应用程序定义 CI/CD 工作流。

Pipeline 资源的定义由多个字段或属性组成，它们一起可让管道实现一个特定目标。每个 **Pipeline** 资源定义必须至少包含一个**Task（任务）**资源，用于控制特定输入并生成特定的输出。Pipeline 定义也可以根据应用程序要求包括 *Conditions*、*Workspaces*、*Parameters* 或 *Resources*。

以下示例显示了 **build-and-deploy** pipeline，它使用 **buildah ClusterTask** 资源从 Git 存储库构建应用程序镜像：

```
apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "pipelines-1.10"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks: 7
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
```

```
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image 8
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests 9
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter: 10
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

1. Pipeline API 版本 **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中, **Pipeline**。
3. 此 Pipeline 的唯一名称。
4. 指定 Pipeline 的定义和结构。

5 Pipeline 中所有任务使用的工作区。

6 Pipeline 中所有任务使用的参数。

7 指定 Pipeline 中使用的任务列表。

8 任务 **build-image** 使用 **buildah** ClusterTask 从给定的 Git 仓库构建应用程序镜像。

9 任务 **apply-manifests** 使用相同名称的用户定义的任务。

10 指定在 Pipeline 中运行任务的顺序。在本例中，**apply-manifests** 任务仅在 **build-image** 任务完成后运行。

注意

Red Hat OpenShift Pipelines Operator 安装 Buildah 集群任务，并创建具有足够权限来构建和推送镜像的**管道**服务帐户。当与没有权限不足的不同服务帐户关联时，Buildah 集群任务可能会失败。

4.2.2.6. PipelineRun

PipelineRun 是一种资源类型，它绑定了管道、工作区、凭证和一组特定于运行 CI/CD 工作流的情况的参数值。

*管道运行（pipeline run）*是管道的运行实例。它使用集群上的特定输入、输出和执行参数来实例化 Pipeline 执行。它还为管道运行中的每个任务创建一个任务运行。

管道按顺序运行任务，直到任务完成或任务失败为止。**status** 字段跟踪和每个任务运行的进度，并存储它以用于监控和审计目的。

以下示例使用相关的资源和参数运行 **build-and-deploy** 管道：

```
apiVersion: tekton.dev/v1beta1 1
kind: PipelineRun 2
metadata:
  name: build-deploy-api-pipelinerun 3
spec:
  pipelineRef:
    name: build-and-deploy 4
  params: 5
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: https://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
```

```
    resources:
      requests:
        storage: 500Mi
```

1. Pipeline 运行 API 版本 **v1beta1**。
2. Kubernetes 对象的类型。在本例中，**PipelineRun**。
3. 用于标识此管道运行的唯一名称。
4. 要运行的管道的名称。在本例中，**build-and-deploy**。
5. 运行管道所需的参数列表。
6. 管道运行使用的工作区。

其他资源

- 使用 git secret 验证管道

4.2.2.7. Workspaces（工作区）

注意

建议您在 OpenShift Pipelines 中使用 Workspaces 而不是 PipelineResources，因为 PipelineResources 很难调试，范围有限，且不容易重复使用。

Workspace 声明 Pipeline 中的任务在运行时需要的共享存储卷来接收输入或提供输出。Workspaces 不指定卷的实际位置，它允许您定义运行时所需的文件系统或部分文件系统。Task 或 Pipeline 会声明 Workspace，您必须提供卷的特定位置详情。然后，它会挂载到 TaskRun 或 PipelineRun 中的 Workspace 中。这种将卷声明与运行时存储卷分开来使得任务可以被重复使用、灵活且独立于用户环境。

使用 Workspaces，您可以：

- 存储任务输入和输出
- 任务间共享数据
- 使用它作为 Secret 中持有的凭证的挂载点
- 使用它作为 ConfigMap 中保存的配置的挂载点
- 使用它作为机构共享的通用工具的挂载点
- 创建可加快作业的构建工件缓存

您可以使用以下方法在 TaskRun 或 PipelineRun 中指定 Workspaces:

- 只读 ConfigMap 或 Secret
- 与其他任务共享的现有 PersistentVolumeClaim
- 来自提供的 VolumeClaimTemplate 的 PersistentVolumeClaim
- TaskRun 完成后丢弃的 emptyDir

以下显示了 **build-and-deploy** Pipeline 的代码片段，它为任务 **build-image** 和 **apply-manifests** 声明了一个 **shared-workspace** Workspace。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces: 1
  - name: shared-workspace
  params:
...
  tasks: 2
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces: 3
    - name: source 4
      workspace: shared-workspace 5
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: 6
    - name: source
      workspace: shared-workspace
    runAfter:
      - build-image
...
```

1. Pipeline 中定义的任务共享的 Workspace 列表。Pipeline 可以根据需要定义 Workspace。在这个示例中，只声明了一个名为 **shared-workspace** 的 Workspace。
2. Pipeline 中使用的任务定义。此片段定义了两个任务，**build-image** 和 **apply-manifests**。这两个任务共享一个 Workspace。
3. **build-image** 任务中使用的 Workspaces 列表。任务定义可以根据需要包含多个 Workspace。但建议任务最多使用一个可写 Workspace。
4. 唯一标识任务中使用的 Workspace 的名称。此任务使用一个名为 **source** 的 Workspace。
5. 任务使用的 Pipeline Workspace 的名称。请注意，Workspace **source** 使用 Pipeline Workspace **shared-workspace**。
6. **apply-manifests** 任务中使用的 Workspace 列表。请注意，此任务与 **build-image** 任务共享 **source** Workspace。

工作区可帮助任务共享数据，并允许您指定 Pipeline 中每个任务在执行过程中所需的一个或多个卷。您可以创建持久性卷声明，或者提供一个卷声明模板，用于为您创建持久性卷声明。

以下 **build-deploy-api-pipelinerun** PipelineRun 的代码片段使用卷声明模板创建持久性卷声明来为 **build-and-deploy** Pipeline 中使用的 **shared-workspace** Workspace 定义存储卷。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
  params:
...

  workspaces: 1
  - name: shared-workspace 2
    volumeClaimTemplate: 3
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

1. 指定 Pipeline Workspaces 列表，用于在 PipelineRun 中提供卷绑定。
2. 提供卷的 Pipeline 中的 Workspace 的名称。
3. 指定卷声明模板，该模板可创建一个持久性卷声明来为工作区定义存储卷。

4.2.2.8. 触发器

使用*触发器（Trigger）*和 Pipelines 一起创建一个完整的 CI/CD 系统，其中 Kubernetes 资源定义整个 CI/CD 执行。触发器捕获外部事件，如 Git 拉取请求，并处理它们以获取关键信息。将这个事件数据映射到一组预定义的参数会触发一系列任务，然后创建和部署 Kubernetes 资源并实例化管道。

例如，您可以使用 Red Hat OpenShift Pipelines 为应用程序定义 CI/CD 工作流。管道必须启动，才能在应用程序存储库中使任何新的更改生效。通过捕获和处理任何更改事件，并通过触发器部署新镜像的管道运行来自动触发这个过程。

触发器由以下主要资源组成，它们可一起组成可重复使用、分离和自力更生的 CI/CD 系统：

- **TriggerBinding** 资源从事件有效负载中提取字段，并将它们保存为参数。
  以下示例显示了 **TriggerBinding** 资源的代码片段,它从接收的事件有效负载中提取 Git 存储库信息：

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: TriggerBinding 2
metadata:
  name: vote-app 3
spec:
  params: 4
```

```
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

1. **TriggerBinding** 资源的 API 版本。在本例中，**v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TriggerBinding**。
3. 用于标识 **TriggerBinding** 资源的唯一名称。
4. 从接收的事件有效负载中提取并传递给 **TriggerTemplate** 的参数列表。在本例中，Git 仓库 URL、名称和修订版本是从事件有效负载主体中提取的。

- **TriggerTemplate** 资源充当创建资源的方式标准。它指定了 **TriggerBinding** 资源中参数化数据的方式。触发器模板从触发器绑定接收输入，然后执行一系列操作来创建新管道资源，并启动新管道运行。
  以下示例显示了 **TriggerTemplate** 资源的代码片段，它使用您刚创建的 **TriggerBinding** 资源提供的 Git 存储库信息创建一个管道运行：

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: TriggerTemplate 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.10
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(tt.params.git-repo-name)-$(uid)
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-registry.svc:5000/pipelines-
```

```
tutorial/$(tt.params.git-repo-name)
    workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
       spec:
        accessModes:
         - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

1. **TriggerTemplate** 资源的 API 版本。在本例中，**v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中，**TriggerTemplate**。
3. 用于标识 **TriggerTemplate** 资源的唯一名称。
4. **TriggerBinding** 资源提供的参数。
5. 指定使用 **TriggerBinding** 或 **EventListener** 资源接收的参数创建资源方法的模板列表。

- **Trigger** 资源组合了 **TriggerBinding** 和 **TriggerTemplate** 资源，以及可选的 **interceptors** 事件处理器。
  拦截器会处理在 **TriggerBinding** 资源之前运行的特定平台的所有事件。您可以使用拦截器过滤载荷，验证事件，定义和测试触发器条件，以及实施其他有用的处理。拦截器使用 secret 进行事件验证。在事件数据穿过拦截器后，在将有效负载数据传递给触发器之前，它会被发送到触发器。您还可以使用拦截器修改 **EventListener** 规格中引用的关联触发器的行为。

  以下示例显示了一个 **Trigger** 资源的代码片段，名为 **vote-trigger**，它连接 **TriggerBinding** 和 **TriggerTemplate** 资源，以及 **interceptors** 事件处理器。

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: Trigger 2
metadata:
  name: vote-trigger 3
spec:
  serviceAccountName: pipeline 4
  interceptors:
    - ref:
        name: "github" 5
      params: 6
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
    - ref: vote-app 7
  template: 8
     ref: vote-app
---
apiVersion: v1
kind: Secret 9
```

```
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "1234567"
```

1. **Trigger** 资源的 API 版本。在本例中， **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中, **Trigger**。
3. 用于标识 **Trigger** 资源的唯一名称。
4. 要使用的服务帐户名称。
5. 要被引用的拦截器名称。在本例中，是 **github**。
6. 要指定的参数.
7. 连接到 **TriggerTemplate** 资源的 **TriggerBinding** 资源的名称。
8. 连接到 **TriggerBinding** 资源的 **TriggerTemplate** 资源的名称。
9. 用于验证事件的 Secret。

- **EventListener** 资源提供一个端点或事件接收器（sink），用于使用 JSON 有效负载侦听传入的基于 HTTP 的事件。它从每个 **TriggerBinding** 资源提取事件参数，然后处理此数据以按照对应的 **TriggerTemplate** 资源指定的 Kubernetes 资源创建 Kubernetes 资源。 **EventListener** 资源还使用事件 **interceptors（拦截器）** 在有效负载上执行轻量级事件处理或基本过滤，这可识别有效负载类型并进行自选修改。目前，管道触发器支持五种拦截器：*Webhook Interceptors*, *GitHub 拦截器*、*GitLab 拦截器*、*Bitbucket 拦截器* 和 *Common Expression Language (CEL) 拦截器*。以下示例显示了一个 **EventListener** 资源，它引用名为 **vote-trigger** 的 **Trigger** 资源。

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: EventListener 2
metadata:
  name: vote-app 3
spec:
  serviceAccountName: pipeline 4
  triggers:
    - triggerRef: vote-trigger 5
```

1. **EventListener** 资源的 API 版本。在本例中， **v1beta1**。
2. 指定 Kubernetes 对象的类型。在本例中， **EventListener**。
3. 用于标识 **EventListener** 资源的唯一名称。
4. 要使用的服务帐户名称。
5. **EventListener** 资源引用的 **Trigger** 资源的名称。

4.2.3. 其他资源

- 有关安装管道的详情，请参阅安装 OpenShift Pipelines。

- 有关创建自定义 CI/CD 解决方案的详情，请参阅使用 CI/CD Pipelines 创建应用程序。
- 有关重新加密 TLS 终止的详情，请参阅重新加密终止。
- 有关安全路由的详情，请参阅安全路由部分。

4.3. 安装 OPENSHIFT PIPELINES

本指南帮助集群管理员了解将 Red Hat OpenShift Pipelines Operator 安装到 OpenShift Container Platform 集群的整个过程。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 已安装了 **oc** CLI。
- 您已在本地系统中安装了 OpenShift Pipelines (tkn)CLI。

4.3.1. 在 Web 控制台中安装 Red Hat OpenShift Pipelines Operator

您可以使用 OpenShift Container Platform OperatorHub 中列出的 Operator 来安装 Red Hat OpenShift Pipelines。安装 Red Hat OpenShift Pipelines Operator 时，管道配置所需的自定义资源（CR）与 Operator 一起自动安装。

默认 Operator 自定义资源定义（CRD）**config.operator.tekton.dev** 现在被 **tektonconfigs.operator.tekton.dev** 替代。另外，Operator 提供以下额外的 CRD 来单独管理 OpenShift Pipelines 组
件：**tektonpipelines.operator.tekton.dev**、**tektontriggers.operator.tekton.dev** 和 **tektonaddons.operator.tekton.dev**。

如果在集群中安装了 OpenShift Pipelines，现有安装会无缝升级。Operator 会根据需要将集群中的 **config.operator.tekton.dev** 实例替换为 **tektonconfigs.operator.tekton.dev** 实例，以及其它 CRD 的额外对象。

警告

如果您手动更改现有安装，例如，在 **config.operator.tekton.dev** CRD 实例中修改了目标命名空间（更改了 **resource name - cluster** 的项），则升级过程将不会非常流畅。在这种情况下，推荐的工作流是，先卸载安装，然后再重新安装 Red Hat OpenShift Pipelines Operator。

Red Hat OpenShift Pipelines Operator 现在提供了选择您要安装的组件的选项，方法是作为 **TektonConfig** CR 的一部分来指定配置集。在安装 Operator 时会自动安装 **TektonConfig** CR。支持的配置集有：

- Lite：只安装 Tekton Pipelines。
- Basic：安装 Tekton Pipelines 和 Tekton Triggers。
- All：在安装了 **TektonConfig** CR 时，使用的默认配置集。此配置集安装所有 Tekton 组件：

Tekton Pipelines、Tekton Triggers、Tekton Addons（包括 **ClusterTasks**、**ClusterTriggerBindings**、**ConsoleCLIDownload**、**ConsoleQuickStart** 和 **ConsoleYAMLSample** 资源）。

流程

1. 在控制台的 **Administrator** 视角中，导航到 **Operators → OperatorHub**。
2. 使用 **Filter by keyword** 复选框在目录中搜索 **Red Hat OpenShift Pipelines** Operator。点 **Red Hat OpenShift Pipelines** Operator 标题。
3. 参阅 **Red Hat OpenShift Pipelines** Operator 页中有关 Operator 的简单描述。点 **Install**。
4. 在 **Install Operator** 页面中：
   a. 为 **Installation Mode** 选择 **All namespaces on the cluster (default)**。选择该项会将 Operator 安装至默认**openshift-operators** 命名空间，这将启用 Operator 以进行监视并在集群中的所有命名空间中可用。
   b. 为 **Approval Strategy** 选择 **Automatic**。这样可确保以后对 Operator 的升级由 Operator Lifecycle Manager (OLM) 自动进行。如果您选择 **Manual** 批准策略，OLM 会创建一个更新请求。作为集群管理员，您必须手动批准 OLM 更新请求，才可将 Operator 更新至新版本。
   c. 选择一个 **Update Channel**。
      - **pipelines-<version>** 频道是安装 Red Hat OpenShift Pipelines Operator 的默认频道。例如，安装 Red Hat OpenShift Pipelines Operator 版本 **1.7** 的默认频道是 **pipelines-1.7**。
      - **latest** 频道启用 Red Hat OpenShift Pipelines Operator 最新稳定版本的安装。

注意

preview 和 **stable** 频道将在以后的版本中被弃用并删除。

5. 点 **Install**。您会看到 **Installed Operators** 页面中列出的 Operator。

注意

Operator 会自动安装到 **openshift-operators** 命名空间中。

6. 检查 **Status** 是否已被设置为 **Succeeded Up to date** 来确认 Red Hat OpenShift Pipelines Operator 已安装成功。

警告

即使其他组件的安装正在进行中，成功状态也可能会显示为 **Succeeded Up to date**。因此，在终端中手动验证安装非常重要。

7. 验证 Red Hat OpenShift Pipelines Operator 的所有组件都已成功安装。在终端中登录到集群，并运行以下命令：

```
$ oc get tektonconfig config
```

输出示例

```
NAME     VERSION   READY   REASON
config   1.9.2     True
```

如果 **READY** 条件为 **True**，则代表 Operator 及其组件已被成功安装。

另外，运行以下命令来检查组件版本：

```
$ oc get tektonpipeline,tektontrigger,tektonaddon,pac
```

输出示例

```
NAME                                    VERSION   READY   REASON
tektonpipeline.operator.tekton.dev/pipeline   v0.41.1   True
NAME                                   VERSION   READY   REASON
tektontrigger.operator.tekton.dev/trigger   v0.22.2   True
NAME                                VERSION   READY   REASON
tektonaddon.operator.tekton.dev/addon   1.9.2     True
NAME                                                VERSION   READY   REASON
openshiftpipelinesascode.operator.tekton.dev/pipelines-as-code   v0.15.5   True
```

4.3.2. 使用 CLI 安装 OpenShift Pipelines Operator

您可以使用 CLI 从 OperatorHub 安装 Red Hat OpenShift Pipelines Operator。

流程

1. 创建一个订阅对象 YAML 文件，以便为 Red Hat OpenShift Pipelines Operator 订阅一个命名空间，如 **sub.yaml**：

订阅示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel:  <channel name> 1
  name: openshift-pipelines-operator-rh 2
  source: redhat-operators 3
  sourceNamespace: openshift-marketplace 4
```

1. Operator 的频道名称。**pipelines-<version>** 频道是默认频道。例如，Red Hat OpenShift Pipelines Operator 版本 **1.7** 的默认频道是 **pipelines-1.7**。**latest** 频道启用 Red Hat OpenShift Pipelines Operator 最新稳定版本的安装。

2 要订阅的 Operator 的名称。

3 提供 Operator 的 CatalogSource 的名称。

4 CatalogSource 的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub CatalogSource。

2. 创建订阅对象：

```
$ oc apply -f sub.yaml
```

Red Hat OpenShift Pipelines Operator 现在安装在默认目标命名空间 **openshift-operators** 中。

4.3.3. 在受限环境中的 Red Hat OpenShift Pipelines Operator

Red Hat OpenShift Pipelines Operator 支持在受限网络环境中安装管道。

Operator 会安装代理 Webhook，它会根据 **cluster** 代理对象在由 tekton-controllers 创建的 pod 容器中设置代理环境变量。它还会在 **TektonPipelines**、**TektonTriggers**、**Controllers**、**Webhooks** 和 **Operator Proxy Webhook** 资源中设置代理环境变量。

默认情况下，**openshift-pipelines** 命名空间禁用代理 Webhook。要为任何其他命名空间禁用它，您可以在 **namespace** 对象中添加 **operator.tekton.dev/disable-proxy: true** 标签。

4.3.4. 其他资源

- 您可以参阅将 Operators 添加到集群一节中的内容来了解更多有关在 OpenShift Container Platform 上安装 Operator 的信息。
- 要使用 Red Hat OpenShift Pipelines Operator 安装 Tekton 链，请参阅为 Red Hat OpenShift Pipelines 提供链安全 使用 Tekton 链。
- 要安装和部署 in-cluster Tekton Hub，请参阅 使用带有 Red Hat OpenShift Pipelines 的 Tekton Hub。
- 有关在受限环境中使用管道的更多信息，请参阅：
  - 镜像以在受限环境中运行管道
  - 为受限集群配置 Samples Operator
  - 创建带有镜像 registry 的集群

4.4. 卸载 OPENSHIFT PIPELINES

集群管理员可以通过执行以下步骤卸载 Red Hat OpenShift Pipelines Operator：

1. 删除安装 Red Hat OpenShift Pipelines Operator 时默认添加的自定义资源 (CR)。
2. 删除依赖于 Operator 的可选组件的 CR，如 Tekton Hub。

小心

如果在不删除可选组件的 CR 的情况下卸载 Operator，则无法在以后删除它们。

3. 卸载 Red Hat OpenShift Pipelines Operator。

安装 Operator 时，仅卸载 Operator 不会删除默认创建的 Red Hat OpenShift Pipelines 组件。

4.4.1. 删除 Red Hat OpenShift Pipelines 组件和自定义资源

删除安装 Red Hat OpenShift Pipelines Operator 期间默认创建的自定义资源（CR）。

流程

1. 在 Web 控制台的 **Administrator** 视角中，导航至 **Administration → Custom Resource Definition**。
2. 在 **Filter by name** 框中键入 **config.operator.tekton.dev** 来搜索 Red Hat OpenShift Pipelines Operator CR。
3. 点击 **CRD Config** 查看 **Custom Resource Definition Details** 页面。
4. 点击 **Actions** 下拉菜单并选择 **Delete Custom Resource Definition**。

注意

删除 CR 将删除 Red Hat OpenShift Pipelines 组件，并丢失集群上的所有任务和管道。

5. 点击 **Delete** 以确认删除 CR。

重要

在卸载 Operator 前，重复查找和删除可选组件的 CR，如 Tekton Hub。如果在不删除可选组件的 CR 的情况下卸载 Operator，则无法在以后删除它们。

4.4.2. 卸载 Red Hat OpenShift Pipelines Operator

您可以使用 web 控制台中的 **Administrator** 视角卸载 Red Hat OpenShift Pipelines Operator。

流程

1. 在 **Operators → OperatorHub** 页面中，使用 **Filter by keyword** 复选框来搜索 **Red Hat OpenShift Pipelines** Operator。
2. 点 **Red Hat OpenShift Pipelines** Operator 标题。Operator 标题表示已安装 Operator。
3. 在 **Red Hat OpenShift Pipelines** Operator 描述页面中，点 **Uninstall**。

其他资源

- 您可以参阅从集群中卸载 Operators一节中的内容来了解更多有关从 OpenShift Container Platform 上卸载 Operator 的信息。

4.5. 为使用 OPENSHIFT PIPELINES 的应用程序创建 CI/CD 解决方案

使用 Red Hat OpenShift Pipelines，您可以创建一个自定义的 CI/CD 解决方案来构建、测试和部署应用程序。

要为应用程序创建一个完整的自助 CI/CD 管道，请执行以下任务：

- 创建自定义任务，或安装现有的可重复使用的任务。
- 为应用程序创建并定义交付管道。
- 使用以下方法之一提供附加到管道执行的工作区中的存储卷或文件系统：
  - 指定创建持久性卷声明的卷声明模板
  - 指定一个持久性卷声明
- 创建一个 **PipelineRun** 对象来实例化并调用管道。
- 添加触发器以捕获源仓库中的事件。

本节使用 **pipelines-tutorial** 示例来演示前面的任务。这个示例使用一个简单的应用程序，它由以下部分组成：

- 一个前端接口，**pipelines-vote-ui**，它的源代码在 **pipelines-vote-ui** Git 存储库中。
- 一个后端接口 **pipelines-vote-api**，它的源代码在 **pipelines-vote-api** Git 存储库中。
- **apply-manifests** 和 **update-deployment** 任务在 **pipelines-tutorial** Git 存储库中。

4.5.1. 先决条件

- 有访问 OpenShift Container Platform 集群的权限。
- 已使用在 OpenShift OperatorHub 中列出的 Red Hat OpenShift Pipelines Operator 安装了 OpenShift Pipelines。在安装后，它可用于整个集群。
- 已安装 OpenShift Pipelines CLI。
- 使用您的 GitHub ID fork 前端 **pipelines-vote-ui** 和后端 **pipelines-vote-api** Git 存储库，并具有对这些存储库的管理员访问权限。
- 可选：已克隆了 **pipelines-tutorial** Git 存储库。

4.5.2. 创建项目并检查管道服务帐户

流程

1. 登录您的 OpenShift Container Platform 集群：

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. 为示例应用程序创建一个项目。在本例中，创建 **pipelines-tutorial** 项目：

```
$ oc new-project pipelines-tutorial
```

注意

如果您使用其他名称创建项目，请确定使用您的项目名称更新示例中使用的资源 URL。

3. 查看 **pipeline** 服务帐户：
Red Hat OpenShift Pipelines Operator 添加并配置一个名为 **pipeline** 的服务帐户，该帐户有足够的权限来构建和推送镜像。**PipelineRun** 对象使用此服务帐户。

```
$ oc get serviceaccount pipeline
```

4.5.3. 创建管道任务

流程

1. 从 **pipelines-tutorial** 存储库安装 **apply-manifests** 和 **update-deployment** 任务资源，其中包含可为管道重复使用的任务列表：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/02_update_deployment_task.yaml
```

2. 使用 **tkn task list** 命令列出您创建的任务：

```
$ tkn task list
```

输出会确认创建了 **apply-manifests** 和 **update-deployment** 任务：

```
NAME                DESCRIPTION   AGE
apply-manifests                   1 minute ago
update-deployment                 48 seconds ago
```

3. 使用 **tkn clustertasks list** 命令列出由 Operator 安装的额外集群任务，如 **buildah** 和 **s2i-python**：

注意

要在受限环境中使用 **buildah** 集群任务，您必须确保 Dockerfile 使用内部镜像流作为基础镜像。

```
$ tkn clustertasks list
```

输出列出了 Operator 安装的 **ClusterTask** 资源：

```
NAME                DESCRIPTION   AGE
buildah                           1 day ago
git-clone                         1 day ago
s2i-python                        1 day ago
tkn                               1 day ago
```

其他资源

- 管理未指定版本的和版本化的集群任务

4.5.4. 组装管道

管道（pipeline）代表一个 CI/CD 流，由要执行的任务定义。它被设计为在多个应用程序和环境中通用且可重复使用。

管道指定任务如何使用 **from** 和 **runAfter** 参数相互交互以及它们执行的顺序。它使用 **workspaces** 字段指定管道中每个任务在执行过程中所需的一个或多个卷。

在本小节中，您将创建一个管道，从 GitHub 获取应用程序的源代码，然后在 OpenShift Container Platform 上构建和部署应用程序。

管道为后端应用程序 **pipelines-vote-api** 和前端应用程序 **pipelines-vote-ui** 执行以下任务：

- 通过引用 **git-url** 和 **git-revision** 参数，从 Git 存储库中克隆应用程序的源代码。
- 使用 **buildah** 集群任务构建容器镜像。
- 通过引用 **image** 参数将镜像推送到 OpenShift 镜像 registry。
- 通过使用 **apply-manifests** 和 **update-deployment** 任务在 OpenShift Container Platform 上部署新镜像。

流程

1. 复制以下管道 YAML 文件示例内容并保存：

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
  - name: shared-workspace
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "pipelines-1.10"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
```

```
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

Pipeline 定义提取 Git 源存储库和镜像 registry 的特定内容。当一个管道被触发并执行时，这些详细信息会作为 **params** 添加。

2. 创建管道：

```
$ oc create -f <pipeline-yaml-file-name.yaml>
```

或者，还可以从 Git 存储库直接执行 YAML 文件：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/04_pipeline.yaml
```

3. 使用 **tkn pipeline list** 命令来验证管道是否已添加到应用程序中：

```
$ tkn pipeline list
```

检查输出来验证创建了 **build-and-deploy** pipeline：

```
NAME             AGE           LAST RUN   STARTED   DURATION   STATUS
build-and-deploy   1 minute ago   ---        ---       ---        ---
```

4.5.5. 镜像以在受限环境中运行管道

要在断开连接的集群或受限环境中置备的集群中运行 OpenShift Pipelines，请确保为受限网络配置了 Samples Operator，或者集群管理员创建了带有镜像 registry 的集群。

以下流程使用 **pipelines-tutorial** 示例，使用带有镜像 registry 的集群在受限环境中为应用程序创建管道。为确保 **pipelines-tutorial** 示例在受限环境中工作，您必须为前端接口（**pipelines-vote-ui**）后端接口（**pipelines-vote-api**）和 **cli** 从 mirror registry 中镜像相应的构建器镜像。

流程

1. 为前端接口 **pipelines-vote-ui** 从 mirror registry 中镜像构建器镜像。

   a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream python -n openshift
```

   输出示例

```
Name:   python
Namespace:  openshift
[...]

3.8-ubi8 (latest)
  tagged from registry.redhat.io/ubi8/python-38:latest
    prefer registry pullthrough when referencing this tag

  Build and run Python 3.8 applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-python-container/blob/master/3.8/README.md.
  Tags: builder, python
  Supports: python:3.8, python
  Example Repo: https://github.com/sclorg/django-ex.git

[...]
```

   b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror registry.redhat.io/ubi8/python-38:latest <mirror-registry>:<port>/ubi8/python-38
```

   c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/ubi8/python-38 python:latest --scheduled -n openshift
```

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream python -n openshift
```

输出示例

```
Name:   python
Namespace:  openshift
[...]

latest
  updates automatically from registry <mirror-registry>:<port>/ubi8/python-38

  * <mirror-registry>:<port>/ubi8/python-
38@sha256:3ee3c2e70251e75bfeac25c0c33356add9cc4abcbc9c51d858f39e4dc29c5f58


[...]
```

2. 为后端接口 **pipelines-vote-api** 从 mirror registry 中镜像构建器镜像。

a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream golang -n openshift
```

输出示例

```
Name:   golang
Namespace:  openshift
[...]

1.14.7-ubi8 (latest)
  tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
    prefer registry pullthrough when referencing this tag

  Build and run Go applications on UBI 8. For more information about using this builder
image, including OpenShift considerations, see https://github.com/sclorg/golang-
container/blob/master/README.md.
  Tags: builder, golang, go
  Supports: golang
  Example Repo: https://github.com/sclorg/golang-ex.git

[...]
```

b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror registry.redhat.io/ubi8/go-toolset:1.14.7 <mirror-registry>:
<port>/ubi8/go-toolset
```

c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/ubi8/go-toolset golang:latest --scheduled -n openshift
```

-

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream golang -n openshift
```

输出示例

```
Name:   golang
Namespace:  openshift
[...]

latest
  updates automatically from registry <mirror-registry>:<port>/ubi8/go-toolset

  * <mirror-registry>:<port>/ubi8/go-
toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421
b37

[...]
```

3. 从 **cli** 的镜像 registry 中镜像构建器镜像。

a. 验证所需镜像标签没有导入：

```
$ oc describe imagestream cli -n openshift
```

输出示例

```
Name:                cli
Namespace:              openshift
[...]

latest
  updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


  * quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

b. 将支持的镜像标签镜像到私有 registry：

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
<mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

c. 导入镜像：

```
$ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest --
scheduled -n openshift
```

您必须定期重新导入镜像。**--scheduled** 标志启用镜像自动重新导入。

d. 验证带有指定标签的镜像已被导入：

```
$ oc describe imagestream cli -n openshift
```

输出示例

```
Name:                   cli
Namespace:              openshift
[...]

latest
  updates automatically from registry <mirror-registry>:<port>/openshift-release-dev/ocp-
v4.0-art-dev

  * <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

其他资源

- 为受限集群配置 Samples Operator
- 创建带有镜像 registry 的集群

4.5.6. 运行管道

PipelineRun 资源启动管道，并将其与 Git 和用于特定调用的镜像资源相关联。它为管道中的每个任务自动创建并启动 **TaskRun** 资源。

流程

1. 启动后端应用程序的管道：

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-api \
    -p git-url=https://github.com/openshift/pipelines-vote-api.git \
    -p IMAGE='image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/pipelines-vote-api' \
    --use-param-defaults
```

上一命令使用卷声明模板，该模板为管道执行创建持久性卷声明。

2. 要跟踪管道运行的进度，请输入以下命令：

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上述命令中的 <pipelinerun_id> 是上一命令输出返回的 **PipelineRun** 的 ID。

3. 启动前端应用程序的管道：

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-ui \
    -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
    -p IMAGE='image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/pipelines-vote-ui' \
    --use-param-defaults
```

4. 要跟踪管道运行的进度，请输入以下命令：

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上述命令中的 <pipelinerun_id> 是上一命令输出返回的 **PipelineRun** 的 ID。

5. 几分钟后，使用 **tkn pipelinerun list** 命令列出所有管道运行来验证管道是否成功运行：

```
$ tkn pipelinerun list
```

输出列出了管道运行：

```
NAME                     STARTED      DURATION     STATUS
build-and-deploy-run-xy7rw   1 hour ago   2 minutes    Succeeded
build-and-deploy-run-z2rz8   1 hour ago   19 minutes   Succeeded
```

6. 获取应用程序路由：

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

记录上一个命令的输出。您可以使用此路由来访问应用程序。

7. 要重新运行最后的管道运行,请使用上一管道的管道资源和服务帐户运行：

```
$ tkn pipeline start build-and-deploy --last
```

其他资源

- 使用 git secret 验证管道

4.5.7. 在管道中添加触发器

触发器（Trigger）使 Pipelines 可以响应外部 GitHub 事件，如推送事件和拉取请求。在为应用程序组装并启动管道后，添加 **TriggerBinding**、**TriggerTemplate**、**Trigger** 和 **EventListener** 资源来捕获 GitHub 事件。

流程

1. 复制以下 **TriggerBinding** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

2. 创建 **TriggerBinding** 资源：

```
$ oc create -f <triggerbinding-yaml-file-name.yaml>
```

或者，您可以直接从 **pipelines-tutorial** Git 仓库创建 **TriggerBinding** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/03_triggers/01_binding.yaml
```

3. 复制以下 **TriggerTemplate** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.10
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      generateName: build-deploy-$(tt.params.git-repo-name)-
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
```

```
      value: $(tt.params.git-revision)
    - name: IMAGE
      value: image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/$(tt.params.git-repo-name)
  workspaces:
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

模板指定一个卷声明模板，用于创建用于为工作空间定义存储卷的持久性卷声明。因此，您不需要创建持久性卷声明来提供数据存储。

4. 创建 **TriggerTemplate** 资源：

```
$ oc create -f <triggertemplate-yaml-file-name.yaml>
```

另外，您还可以从 **pipelines-tutorial** Git 仓库直接创建 **TriggerTemplate** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-
1.10/03_triggers/02_template.yaml
```

5. 复制以下 **Trigger** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: vote-trigger
spec:
  serviceAccountName: pipeline
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
```

6. 创建 **Trigger** 资源：

```
$ oc create -f <trigger-yaml-file-name.yaml>
```

另外，您还可以直接从 **pipelines-tutorial** Git 仓库创建 **Trigger** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-
1.10/03_triggers/03_trigger.yaml
```

7. 复制以下 **EventListener** YAML 示例文件的内容并保存：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
```

```
spec:
  serviceAccountName: pipeline
  triggers:
    - triggerRef: vote-trigger
```

或者，如果您还没有定义触发器自定义资源，将绑定和模板规格添加到 **EventListener** YAML 文件中，而不是引用触发器的名称：

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
    template:
      ref: vote-app
```

8. 通过执行以下步骤来创建 **EventListener** 资源：

- 使用安全 HTTPS 连接创建 **EventListener** 资源：

a. 添加一个标签，在 Eventlistener 资源中启用安全 **HTTPS** 连接：

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

b. 创建 **EventListener** 资源：

```
$ oc create -f <eventlistener-yaml-file-name.yaml>
```

或者，您可以直接从 **pipelines-tutorial** Git 仓库创建 **EvenListener** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/03_triggers/04_event_listener.yaml
```

c. 使用重新加密 TLS 终止创建路由：

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

另外，您可以创建一个重新加密 TLS 终止 YAML 文件，以创建安全路由。

安全路由重新加密 TLS 终止 YAML 示例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: <hostname>
  to:
    kind: Service
```

```
    name: frontend (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1) (2) 对象的名称，长度限于 63 个字符。

(3) **termination** 字段设置为 **reencrypt**。这是唯一需要 **tls** 的字段。

(4) 重新加密需要。**destinationCACertificate** 指定用来验证端点证书的 CA 证书，保护从路由器到目标 pod 的连接。如果服务使用服务签名证书，或者管理员为路由器指定默认 CA 证书，且服务有由该 CA 签名的证书，则可以省略此字段。

如需了解更多选项，请参阅 **oc create route reencrypt --help**。

- 使用不安全的 HTTP 连接创建 **EventListener** 资源：

  a. 创建 **EventListener** 资源。

  b. 将 **EventListener** 服务公开为 OpenShift Container Platform 路由，使其可以被公开访问：

```
$ oc expose svc el-vote-app
```

4.5.8. 配置事件监听程序为多个命名空间提供服务

注意

如果要创建一个基本的 CI/CD 管道，您可以跳过此部分。但是，如果您的部署策略涉及多个命名空间，您可以将事件监听程序配置为为多个命名空间提供服务。

为了提高 **EvenListener** 对象的可重用性，集群管理员可将它们配置为为多个命名空间的多租户事件监听程序进行配置和部署。

流程

1. 为事件监听程序配置集群范围的获取权限。

   a. 设置在 **ClusterRoleBinding** 和 **EventListener** 对象中使用的服务帐户名称。例如，**el-sa**。

   ServiceAccount.yaml 示例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: el-sa
---
```

b. 在 **ClusterRole.yaml** 文件的 **rules** 部分，为每个事件监听器部署设置适当的权限，以便正常工作集群范围的。

ClusterRole.yaml 示例

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: el-sel-clusterrole
rules:
- apiGroups: ["triggers.tekton.dev"]
  resources: ["eventlisteners", "clustertriggerbindings", "clusterinterceptors",
"triggerbindings", "triggertemplates", "triggers"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["configmaps", "secrets"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["impersonate"]
...
```

c. 使用适当的服务帐户名称和集群角色名称配置集群角色绑定。

ClusterRoleBinding.yaml 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: el-mul-clusterrolebinding
subjects:
- kind: ServiceAccount
  name: el-sa
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: el-sel-clusterrole
...
```

2. 在事件监听器的 **spec** 参数中，添加服务帐户名称，如 **el-sa**。使用事件监听程序要服务的命名空间名称填充 **namespaceSelector** 参数。

EventListener.yaml 示例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: namespace-selector-listener
spec:
  serviceAccountName: el-sa
  namespaceSelector:
    matchNames:
```

```
    - default
    - foo
...
```

3. 创建具有必要权限的服务帐户，如 **foo-trigger-sa**。使用它来绑定触发器。

ServiceAccount.yaml 示例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo-trigger-sa
  namespace: foo
...
```

RoleBinding.yaml 示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggercr-rolebinding
  namespace: foo
subjects:
- kind: ServiceAccount
  name: foo-trigger-sa
  namespace: foo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
...
```

4. 使用适当的触发器模板、触发器绑定和服务帐户名称创建触发器。

Trigger.yaml 示例

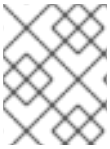
```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  serviceAccountName: foo-trigger-sa
  interceptors:
    - ref:
        name: "github"
      params:
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
```

```
    - ref: vote-app
  template:
    ref: vote-app
...
```

4.5.9. 创建 Webhook

Webhook 是事件监听程序在存储库中配置事件时接收到的 HTTP POST 信息。然后，事件有效负载映射到触发器绑定，并由触发器模板处理。触发器模板最终启动一个或多个管道运行，从而创建并部署 Kubernetes 资源。

在本小节中，您将在 Git 存储库 **pipelines-vote-ui** 和 **pipelines-vote-api** 的副本中配置 webhook URL。这个 URL 指向公开访问的 **EventListener** 服务路由。



注意

添加 Webhook 需要对该存储库有管理特权。如果您没有对库的管理权限，请联络您的系统管理员来添加 webhook。

流程

1. 获取 Webhook URL：
   - 对于安全 HTTPS 连接：

     ```
     $ echo "URL: $(oc  get route el-vote-app --template='https://{{.spec.host}}')"
     ```

   - 对于 HTTP（不安全）连接：

     ```
     $ echo "URL: $(oc  get route el-vote-app --template='http://{{.spec.host}}')"
     ```

     记录下输出中的 URL。
2. 在前端存储库中手动配置 Webhook：
   a. 在浏览器中打开前端 Git 存储库 **pipelines-vote-ui**。
   b. 点 **Settings → Webhooks → Add Webhook**
   c. 在 **Webhooks/Add Webhook** 页面中：
      i. 在 **Payload URL** 字段中输入第 1 步中的 webhook URL
      ii. 为 **Content type** 选择 **application/json**
      iii. 在 **Secret** 字段中指定 secret
      iv. 确定选择了 **Just the push event**
      v. 选择 **Active**
      vi. 点击 **Add webhook**。
3. 重复步骤 2 来使用后端存储库 **pipelines-vote-api**。

4.5.10. 触发一个管道运行

每当 Git 仓库中发生 **push** 事件时，配置的 Webhook 会将事件有效负载发送到公开的 **EventListener** 服务路由。应用程序的 **EventListener** 服务处理有效负载，并将其传递给相关的 **TriggerBinding** 和 **TriggerTemplate** 资源对。**TriggerBinding** 资源提取参数，**TriggerTemplate** 资源使用这些参数并指定必须创建资源的方式。这可能会重建并重新部署应用程序。

在本小节中，您将把一个空的提交推送到前端 **pipelines-vote-ui** 存储库，该存储库将触发管道运行。

流程

1. 在终端中，克隆 fork 的 Git 存储库 **pipelines-vote-ui**：

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.10
```

2. 推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.10
```

3. 检查管道运行是否已触发：

```
$ tkn pipelinerun list
```

请注意，一个新的管道运行被启动。

4.5.11. 为用户定义的项目启用触发器监控事件监听程序

作为集群管理员，在用户定义的项目中收集 **Triggers** 服务的事件监听程序指标，并在 OpenShift Container Platform Web 控制台中显示它们，您可以为每个事件监听程序创建服务监控器。在收到 HTTP 请求时，**Triggers** 服务的事件监听程序返回三个指标数据 - **eventlistener_http_duration_seconds**, **eventlistener_event_count**, 和 **eventlistener_triggered_resources**。

先决条件

- 已登陆到 OpenShift Container Platform Web 控制台。
- 已安装 Red Hat OpenShift Pipelines Operator。
- 您已为用户定义的项目启用了监控。

流程

1. 对于每个事件侦听器，创建一个服务监控器。例如，要查看 **test** 命名空间中的 **github-listener** 事件监听程序的指标，请创建以下服务监控器：

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/managed-by: EventListener
    app.kubernetes.io/part-of: Triggers
    eventlistener: github-listener
  annotations:
    networkoperator.openshift.io/ignore-errors: ""
```

```
  name: el-monitor
  namespace: test
spec:
  endpoints:
    - interval: 10s
      port: http-metrics
  jobLabel: name
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: EventListener
      app.kubernetes.io/part-of: Triggers
      eventlistener: github-listener
...
```

2. 通过将请求发送到事件监听程序来测试服务监控器。例如，推送空提交：

```
$ git commit -m "empty-commit" --allow-empty && git push origin main
```

3. 在 OpenShift Container Platform web 控制台中进入 **Administrator → Observe → Metrics**。

4. 要查看指标，请按名称搜索。例如，若要查看 **github-listener** 事件监听器的 **eventlistener_http_resources** 指标的详细信息，请使用 **eventlistener_http_resources** 关键字搜索。

其他资源

- 为用户定义的项目启用监控

4.5.12. 其他资源

- 要将 Pipelines as Code 和应用程序源代码包含在同一存储库中，请参阅使用 Pipelines as Code。
- 有关在 **Developer** 视角中的管道的更多信息，请参阅 web 控制台部分中的使用管道。
- 要了解更多有关安全性上下文约束（SCC）的信息，请参阅 管理安全性上下文约束部分。
- 如需有关可重复使用的任务的更多示例，请参阅 OpenShift Catalog 仓库。另外，您还可以在 Tekton 项目中看到 Tekton Catalog。
- 要安装并部署 Tekton Hub 的自定义实例，以了解可重新调度的任务和管道，请参阅使用带有 Red Hat OpenShift Pipelines 的 Tekton Hub。
- 有关重新加密 TLS 终止的详情，请参阅重新加密终止。
- 有关安全路由的详情，请参阅安全路由部分。

4.6. 管理未指定版本的和版本化的集群任务

作为集群管理员，安装 Red Hat OpenShift Pipelines Operator 会为每个默认集群任务创建变体，称为 *版本化的集群任务* (VCT) 和 *非版本的集群任务* (NVCT)。例如，安装 Red Hat OpenShift Pipelines Operator v1.7 创建一个 **buildah-1-7-0** VCT 和 **buildah** NVCT。

NVCT 和 VCT 具有相同的元数据、行为和规格，包括 **params**、**workspaces** 和 **steps**。但是，当禁用 Operator 或升级 Operator 时，它们的行为会有所不同。

4.6.1. 非版本和版本的集群任务之间的区别

非版本的和版本化的集群任务有不同的命名约定。另外，Red Hat OpenShift Pipelines Operator 会不同地升级它们。

表 4.5. 非版本和版本的集群任务之间的区别

	非版本的集群任务	版本的集群任务
Nomenclature	NVCT 仅包含集群任务的名称。例如，随 Operator v1.7 安装的 Buildah NVCT 的名称是 **buildah**。	VCT 包含集群任务的名称，后接为后缀的版本。例如，由 Operator v1.7 安装的 Buildah 的 VCT 的名称是 **buildah-1-7-0**。
Upgrade（升级）	升级 Operator 时，它会使用最新的更改更新非版本的集群任务。NVCT 的名称保持不变。	升级 Operator 会安装 VCT 的最新版本并保留更早的版本。VCT 的最新版本对应于升级的 Operator。例如，安装 Operator 1.7 安装 **buildah-1-7-0** 并保留 **buildah-1-6-0**。

4.6.2. 非版本和版本的集群任务的优点和缺陷

在将未指定版本或版本化的集群任务用作生产环境中的标准之前，集群管理员可能会考虑它们的优点和缺点。

表 4.6. 非版本和版本的集群任务的优点和缺陷

集群任务	优点	缺点
非版本的集群任务 (NVCT)	• 如果您希望使用最新更新和程序错误修复部署管道，请使用 NVCT。 • 升级 Operator 会升级非版本的集群任务，这会消耗比多个版本的集群任务更少的资源。	如果您部署使用 NVCT 的管道，如果自动升级的集群任务不向后兼容，则它们可能会在 Operator 升级后中断。

集群任务	优点	缺点
版本化的集群任务 (VCT)	• 如果您的目的是在生产环境中提供稳定的管道，请使用 VCT。 • 之前的版本会保留在集群中，即使安装了集群任务的更新的版本。您可以继续使用较早的集群任务。	• 如果您继续使用集群任务的早期版本，您可能会错过最新的功能和关键安全更新。 • 早期版本的集群任务，它们不正常消耗集群资源。 • * 升级后，Operator 无法管理更早的 VCT。您可以使用 **oc delete clustertask** 命令手动删除更早的 VCT，但您无法恢复它。

4.6.3. 禁用未指定版本和版本的集群任务

作为集群管理员，您可以禁用安装 Pipelines Operator 的集群任务。

流程

1. 要删除所有非版本的集群任务和最新版本的集群任务，请编辑 **TektonConfig** 自定义资源定义 (CRD)并将 **spec.addon.params** 中的 **clusterTasks** 参数设置为 **false**。

   TektonConfig CR 示例

   ```
   apiVersion: operator.tekton.dev/v1alpha1
   kind: TektonConfig
   metadata:
     name: config
   spec:
     params:
     - name: createRbacResource
       value: "false"
     profile: all
     targetNamespace: openshift-pipelines
     addon:
       params:
       - name: clusterTasks
         value: "false"
   ...
   ```

   当您禁用集群任务时，Operator 会删除所有非版本的集群任务，且只从集群中删除最新版本的集群任务。

   注意

   重新启用集群任务将安装非版本的集群任务。

2. 可选： 要删除早期版本的集群任务，请使用以下方法之一：

a. 要删除个别较早版本的集群任务，请使用 **oc delete clustertask** 命令，后面是版本化的集群任务名称。例如：

```
$ oc delete clustertask buildah-1-6-0
```

b. 要删除由旧版本 Operator 创建的所有版本集群任务，您可以删除对应的安装程序设置。例如：

```
$ oc delete tektoninstallerset versioned-clustertask-1-6-k98as
```

小心

如果您删除旧版本的集群任务，则无法恢复它。您只能恢复当前创建的 Operator 版本和未版本的集群任务。

4.7. 在 OPENSHIFT PIPELINES 中使用 TEKTON HUB

重要

Tekton Hub 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Tekton Hub 可帮助您发现、搜索和共享 CI/CD 工作流可重复使用的任务和管道。Tekton Hub 的一个公共实例位于 hub.tekton.dev 中。集群管理员也可以安装和部署 Tekton Hub 的自定义实例供企业用户使用。

4.7.1. 在 OpenShift Container Platform 集群上安装并部署 Tekton Hub

Tekton Hub 是一个可选组件；集群管理员无法使用 **TektonConfig** 自定义资源(CR)安装它。要安装和管理 Tekton Hub，请使用 **TektonHub** CR。

注意

如果您使用 Github Enterprise 或 Gitlab Enterprise，请在与企业服务器相同的网络中安装并部署 Tekton Hub。例如，如果企业服务器在 VPN 后面运行，请在 VPN 之后也部署 Tekton Hub。

先决条件

- 确保在集群中的默认 **openshift-pipelines** 命名空间中安装了 Red Hat OpenShift Pipelines Operator。

流程

1. 创建 Tekton Hub 仓库的分叉。
2. 克隆已分叉的存储库。
3. 更新 **config.yaml** 文件，使其至少包含一个具有以下范围的用户：

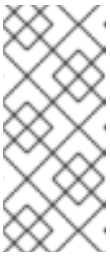
- 具有 **agent:create** 范围，如果目录中有任何更改，可以设置一个 cron 作业，该作业会在间隔后刷新 Tekton Hub 数据库。
- 具有 **catalog:refresh** 范围的用户，它可以刷新目录以及 Tekton Hub 数据库中的所有资源。
- 具有 **config:refresh** 范围的用户，可获取其他范围。

```
...
scopes:
- name: agent:create
  users: <username_registered_with_the_Git_repository_hosting_service_provider>
- name: catalog:refresh
  users: <username_registered_with_the_Git_repository_hosting_service_provider>
- name: config:refresh
  users: <username_registered_with_the_Git_repository_hosting_service_provider>
...
```

支持的服务供应商有 GitHub、GitLab 和 BitBucket。

4. 使用托管供应商的 Git 存储库创建 OAuth 应用程序，并记下客户端 ID 和客户端 Secret。
   - 对于 GitHub OAuth 应用程序，将 **Homepage URL** 和 **Authorization 回调 URL** 设置为 **<auth-route>**。
   - 对于 GitLab OAuth 应用程序，请将 **REDIRECT_URI** 设为 **<auth-route>/auth/gitlab/callback**。
   - 对于 BitBucket OAuth 应用程序，将 **Callback URL** 设置为 **<auth-route>**。
5. 编辑 Tekton Hub API secret 的 **<tekton_hub_repository>/config/02-api/20-api-secret.yaml** 文件中的以下字段：
   - **GH_CLIENT_ID** ：通过 Git 存储库托管服务提供商创建的 OAuth 应用程序的客户端 ID。
   - **GH_CLIENT_SECRET** ：通过 Git 存储库托管服务提供商创建的 OAuth 应用中的客户端 Secret。
   - **GHE_URL** ：GitHub Enterprise URL，如果您要使用 GitHub Enterprise 进行身份验证。不要提供目录的 URL 作为此字段的值。
   - **GL_CLIENT_ID** ：来自 GitLab OAuth 应用程序的客户端 ID。
   - **GL_CLIENT_SECRET** ：来自 GitLab OAuth 应用的客户端 Secret。
   - **GLE_URL**: GitLab Enterprise URL（如果使用 GitLab Enterprise 进行身份验证）。不要提供目录的 URL 作为此字段的值。
   - **BB_CLIENT_ID** ：来自 BitBucket OAuth 应用程序的客户端 ID。
   - **BB_CLIENT_SECRET** ：BitBucket OAuth 应用中的客户端 Secret。
   - **JWT_SIGNING_KEY** ：用于签署为用户创建的 JSON Web 令牌(JWT)的随机字符串。
   - **ACCESS_JWT_EXPIRES_IN** ：添加访问令牌过期的时间限制。例如，**1m**，其中 **m** 表示分钟。支持的时间单位为秒(**s**)、分钟(**m**)、小时(**h**)、天(**d**)和周(**w**)。
   - **REFRESH_JWT_EXPIRES_IN** ：添加刷新令牌过期的时间限制。例如，**1m**，其中 **m** 表示

- **REFRESH_JWT_EXPIRES_IN** ：添加刷新令牌过期的时间限制。例如，**1m**，其中 **m** 表示分钟。支持的时间单位为秒(**s**)、分钟(**m**)、小时(**h**)、天(**d**)和周(**w**)。确保为令牌刷新设置的到期时间大于为令牌访问设置的到期时间。
- **AUTH_BASE_URL** ： OAuth 应用程序的 Route URL。



注意

- 将与客户端 ID 和客户端 Secret 相关的字段用于任何托管服务供应商的 Git 存储库。
- 通过 Git 存储库托管服务提供商注册的帐户凭据，让具有 **catalog: refresh** 的范围可以验证和加载所有目录资源到数据库。

6. 提交更改并将其推送到您的已分叉的存储库。
7. 确保 **TektonHub** CR 类似以下示例：

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonHub
metadata:
  name: hub
spec:
  targetNamespace: openshift-pipelines (1)
  api:
    hubConfigUrl: https://raw.githubusercontent.com/tektoncd/hub/main/config.yaml (2)
```

(1) 需要安装 Tekton Hub 的命名空间；默认为 **openshift-pipelines**。

(2) 使用您 fork 仓库的 **config.yaml** 文件的 URL 替换。

8. 安装 Tekton Hub。

```
$ oc apply -f TektonHub.yaml (1)
```

(1) **TektonConfig** CR 的文件名或路径。

9. 检查安装的状态。

```
$ oc get tektonhub.operator.tekton.dev
NAME   VERSION   READY   REASON   APIURL                 UIURL
hub    v1.7.2    True             https://api.route.url/   https://ui.route.url/
```

4.7.1.1. 在 Tekton Hub 中手动刷新目录

在 OpenShift Container Platform 集群上安装并部署 Tekton Hub 时，也会安装 Postgres 数据库。最初，数据库为空。要将目录中可用的任务和管道添加到数据库，集群管理员必须刷新目录。

先决条件

- 确保您位于 **<tekton_hub_repository>/config/** 目录中。

流程

1. 在 Tekton Hub UI 中，点击 **Login -→ Sign In with GitHub**。

注意

GitHub 在公共的 Tekton Hub UI 中用作示例。对于在集群中的自定义安装，列出了您提供的客户端 ID 和客户端 Secret 的所有 Git 存储库托管服务供应商。

2. 在主页上，点击 user profile 并复制令牌。
3. 调用 Catalog Refresh API。
   - 要使用特定名称刷新目录，请运行以下命令：

```
$ curl -X POST -H "Authorization: <jwt-token>" \ 1
  <api-url>/catalog/<catalog_name>/refresh 2
```

1 从 UI 复制的 Tekton Hub 令牌。

2 目录的 API pod URL 和名称。

输出示例：

```
[{"id":1,"catalogName":"tekton","status":"queued"}]
```

   - 要刷新所有目录，请运行以下命令：

```
$ curl -X POST -H "Authorization: <jwt-token>" \ 1
  <api-url>/catalog/refresh 2
```

1 从 UI 复制的 Tekton Hub 令牌

2 API pod URL。

4. 在浏览器中刷新页面。

4.7.1.2. 可选：设置在 Tekton Hub 中刷新目录的 cron 作业

集群管理员可选择性地设置一个 cron 作业，以便在固定间隔后刷新数据库，因此目录中的更改会出现在 Tekton Hub web 控制台中。

注意

如果资源添加到目录或更新，则刷新目录会在 Tekton Hub UI 中显示这些更改。但是，如果从目录中删除资源，则刷新目录不会从数据库中删除资源。Tekton Hub UI 继续显示已删除的资源。

先决条件

- 确保您位于 **<project_root>/config/** 目录中，其中 **<project_root>** 是克隆的 Tekton Hub 存储库的顶级目录。

- 确保您有一个 JSON Web 令牌(JWT)令牌，其范围有刷新目录。

流程

1. 创建基于代理的 JWT 令牌以供使用。

```
$ curl -X PUT --header "Content-Type: application/json" \
    -H "Authorization: <access-token>" \ 1
    --data '{"name":"catalog-refresh-agent","scopes": ["catalog:refresh"]}' \
    <api-route>/system/user/agent
```

1 JWT 令牌.

带有必要范围的代理令牌以 **{"token":"<agent_jwt_token>"}** 格式返回。请注意返回的令牌，并保留目录刷新 cron 作业。

2. 编辑 **05-catalog-refresh-cj/50-catalog-refresh-secret.yaml** 文件，将 **HUB_TOKEN** 参数设置为上一步中返回的 **<agent_jwt_token>**。

```
apiVersion: v1
kind: Secret
metadata:
  name: catalog-refresh
type: Opaque
stringData:
  HUB_TOKEN: <hub_token> 1
```

1 上一步中返回的 **<agent_jwt_token>**。

3. 应用修改后的 YAML 文件。

```
$ oc apply -f 05-catalog-refresh-cj/ -n openshift-pipelines.
```

4. 可选：默认情况下，cron 任务配置为每 30 分钟运行一次。要更改间隔，修改 **05-catalog-refresh-cj/51-catalog-refresh-cronjob.yaml** 文件中的 **schedule** 参数的值。

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: catalog-refresh
  labels:
    app: tekton-hub-api
spec:
  schedule: "*/30 * * * *"
  ...
```

4.7.1.3. 可选：在 Tekton Hub 配置中添加新用户

流程

1. 根据预期的范围，集群管理员可以在 **config.yaml** 文件中添加新用户。

```
...
scopes:
  - name: agent:create
    users: [<username_1>, <username_2>] (1)
  - name: catalog:refresh
    users: [<username_3>, <username_4>]
  - name: config:refresh
    users: [<username_5>, <username_6>]

default:
  scopes:
    - rating:read
    - rating:write
...
```

(1) 在 Git 存储库托管服务提供商中注册的用户名。

注意

当任何用户首次登录时，它们只会有默认范围，即使它们被添加到 **config.yaml** 中。要激活其他范围，请确保用户至少登录一次。

2. 确保 **config.yaml** 文件中具有 **config-refresh** 范围。

3. 刷新配置。

```
$ curl -X POST -H "Authorization: <access-token>" \ (1)
    --header "Content-Type: application/json" \
    --data '{"force": true} \
    <api-route>/system/config/refresh
```

(1) JWT 令牌.

4.7.2. 在 Developer 视角中选择 Tekton Hub

集群管理员可以选择在 OpenShift Container Platform 集群的 **Developer** 视角的 **Pipeline Builder** 页中显示 Tekton Hub 资源，如任务和管道。

前提条件

- 确保在集群中安装了 Red Hat OpenShift Pipelines Operator，并且 **oc** 命令行工具可用。

流程

- 要在 **Developer** 视角中选择显示 Tekton Hub 资源，将 **TektonConfig** 自定义资源(CR)中的 **enable-devconsole-integration** 字段设置为 **false**。

```
apiVersion: operator.tekton.dev/v1alpha1
  kind: TektonConfig
  metadata:
    name: config
  spec:
```

```
    targetNamespace: openshift-pipelines
    ...
    hub:
      params:
        - name: enable-devconsole-integration
          value: "false"
    ...
```

默认情况下，**TektonConfig** CR 不包括 **enable-devconsole-integration** 字段，Red Hat OpenShift Pipelines Operator 假定值为 **true**。

4.7.3. 其他资源

- Tekton Hub 的 GitHub 存储库。
- 安装 OpenShift Pipelines
- Red Hat OpenShift Pipelines 发行注记

4.8. 使用 PIPELINES AS CODE

使用 Pipelines 作为 Code，具有所需权限的集群管理员和用户可以将管道模板定义为源代码 Git 存储库的一部分。当由源代码推送或配置的 Git 存储库的拉取请求触发时，该功能将运行管道和报告状态。

4.8.1. 主要特性

作为代码的管道支持以下功能：

- 在托管 Git 仓库的平台上拉取请求状态并控制。
- GitHub Checks API 以设置管道运行的状态，包括重新检查。
- GitHub 拉取请求和提交事件。
- 在注释中拉取请求操作，如 **/retest**。
- Git 事件过滤和每个事件一个单独的管道。
- Pipelines 中的自动任务解析，包括本地任务、Tekton Hub 和远程 URL。
- 使用 GitHub blob 和对象 API 检索配置。
- 通过 GitHub 组织的访问控制列表 (ACL)，或使用 Prow 风格的 **OWNER** 文件。
- 用于管理 bootstrap 和 Pipelines 作为代码软件仓库的 **tkn pac** CLI 插件。
- 支持 GitHub App、GitHub Webhook、Bitbucket 服务器和 Bitbucket 云。

4.8.2. 在 OpenShift Container Platform 上安装 Pipelines 作为代码

在安装 Red Hat OpenShift Pipelines Operator 时，默认将安装 Pipelines 作为代码。如果您使用 Pipelines 1.7 或更高版本，请跳过将 Pipelines 作为代码手动安装的步骤。

要使用 Operator 禁用 Pipelines as Code 的默认安装，请在 **TektonConfig** 自定义资源中将 **enable** 参数的值设置为 **false**。

```
...
 spec:
   platforms:
     openshift:
       pipelinesAsCode:
         enable: false
         settings:
           application-name: Pipelines as Code CI
           auto-configure-new-github-repo: "false"
           bitbucket-cloud-check-source-ip: "true"
           hub-catalog-name: tekton
           hub-url: https://api.hub.tekton.dev/v1
           remote-tasks: "true"
           secret-auto-create: "true"
...
```

另外，您可以运行以下命令：

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": false}}}}}'
```

要使用 Red Hat OpenShift Pipelines Operator 启用 Pipelines 的默认安装，在 **TektonConfig** 自定义资源中将 **enable** 参数的值设置为 **true** ：

```
...
spec:
   addon:
     enablePipelinesAsCode: false
...
```

另外，您可以运行以下命令：

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": true}}}}}'
```

4.8.3. 将管道安装为代码 CLI

集群管理员可以在本地机器中使用 **tkn pac** 和 **opc** CLI 工具，或作为容器进行测试。当您为 Red Hat OpenShift Pipelines 安装 **tkn** CLI 时，**tkn pac** 和 **opc** CLI 工具会自动安装。

您可以为支持的平台安装 **tkn pac** 和 **opc** 版本 **1.9.1** 二进制文件：

- Linux (x86_64, amd64)
- Linux on IBM Z and LinuxONE (s390x)
- Linux on IBM Power Systems (ppc64le)
- Mac
- Windows

注意

二进制文件与 **tkn** 版本 **0.23.1** 兼容。

4.8.4. 使用带有 Git 存储库托管服务提供商的 Pipelines as Code。

安装 Pipelines 作为代码后，集群管理员可以配置 Git 存储库托管服务提供商。目前，支持以下服务：

- GitHub 应用程序
- GitHub Webhook
- GitLab
- Bitbucket 服务器
- Bitbucket 云

注意

GitHub App 是与 Pipelines as Code 搭配使用的建议服务。

4.8.5. 使用带有 GitHub 应用程序的 Pipelines as Code

GitHub Apps 充当 Red Hat OpenShift Pipelines 的集点，并为 OpenShift Pipelines 提供了基于 Git 的工作流的优势。集群管理员可以为所有集群用户配置单个 GitHub 应用程序。对于 GitHub Apps 需要使用 Pipelines 作为代码，请确保 GitHub App 的 Webhook 指向 Pipelines，作为代码事件监听器路由（或入口端点）用于侦听 GitHub 事件。

4.8.5.1. 配置 GitHub 应用程序

集群管理员可以通过运行以下命令来创建 GitHub 应用程序：

```
$ tkn pac bootstrap github-app
```

如果没有安装 **tkn pac** CLI 插件，您可以手动创建 GitHub App。

流程

要手动为 Pipelines 作为代码创建和配置 GitHub 应用程序，请执行以下步骤：

1. 登录您的 GitHub 帐户。
2. 进入 **Settings → Developer settings → GitHub Apps**，然后点 **New GitHub App**。
3. 在 GitHub App 表单中提供以下信息：
   - **GitHub 应用程序名称**：**OpenShift Pipelines**
   - **主页 URL** ：OpenShift 控制台 URL
   - **Webhook URL** ：作为代码路由或入口 URL 的 Pipelines。您可以通过运行命令 **echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')** 来找到它。

- **Webhook secret** ：一个任意的机密。您可以通过执行命令 **openssl rand -hex 20** 来生成 secret。

4. 选择以下**仓库权限** ：
   - Checks: **Read & Write**
   - Contents: **Read & Write**
   - Issues: **Read & Write**
   - Metadata: **Read-only**
   - Pull request: **Read & Write**
5. 选择以下**机构权限** ：
   - Members: **Readonly**
   - Plan: **Readonly**
6. 选择以下**用户权限** ：
   - **Commit comment**
   - **Issue comment**
   - **Pull request**
   - **Pull request review**
   - **Pull request review comment**
   - **push**
7. 点 **Create GitHub App**。
8. 在新创建的 GitHub App 的 **Details** 页面中，记录顶部显示的 **App ID**。
9. 在 **Private key** 部分，点 **Generate Private key** 自动生成并下载 GitHub 应用的私钥。安全地存储私钥，以备将来参考和使用。

4.8.5.2. 配置 Pipelines as Code 来访问一个 GitHub 应用程序

要将 Pipelines as Code 配置为访问新创建的 GitHub 应用程序，请执行以下命令：

+

```
$ oc -n openshift-pipelines create secret generic pipelines-as-code-secret \
    --from-literal github-private-key="$(cat <PATH_PRIVATE_KEY>)" \ 1
    --from-literal github-application-id="<APP_ID>" \ 2
    --from-literal webhook.secret="<WEBHOOK_SECRET>" 3
```

1 配置 GitHub 应用程序时下载的私钥的路径。

2 GitHub 应用程序的 **App ID**。

3 创建 GitHub 应用程序时提供的 webhook secret。

注意

Pipelines as Code 通过检测从 GitHub Enterprise 集的标头集并将其用于 GitHub Enterprise API 授权 URL 来自动工作。

4.8.5.3. 在管理员视角中创建 GitHub 应用程序

作为集群管理员，您可以使用 OpenShift Container Platform 集群配置 GitHub 应用程序，以使用 Pipelines as Code。此配置允许您执行构建部署所需的一组任务。

先决条件

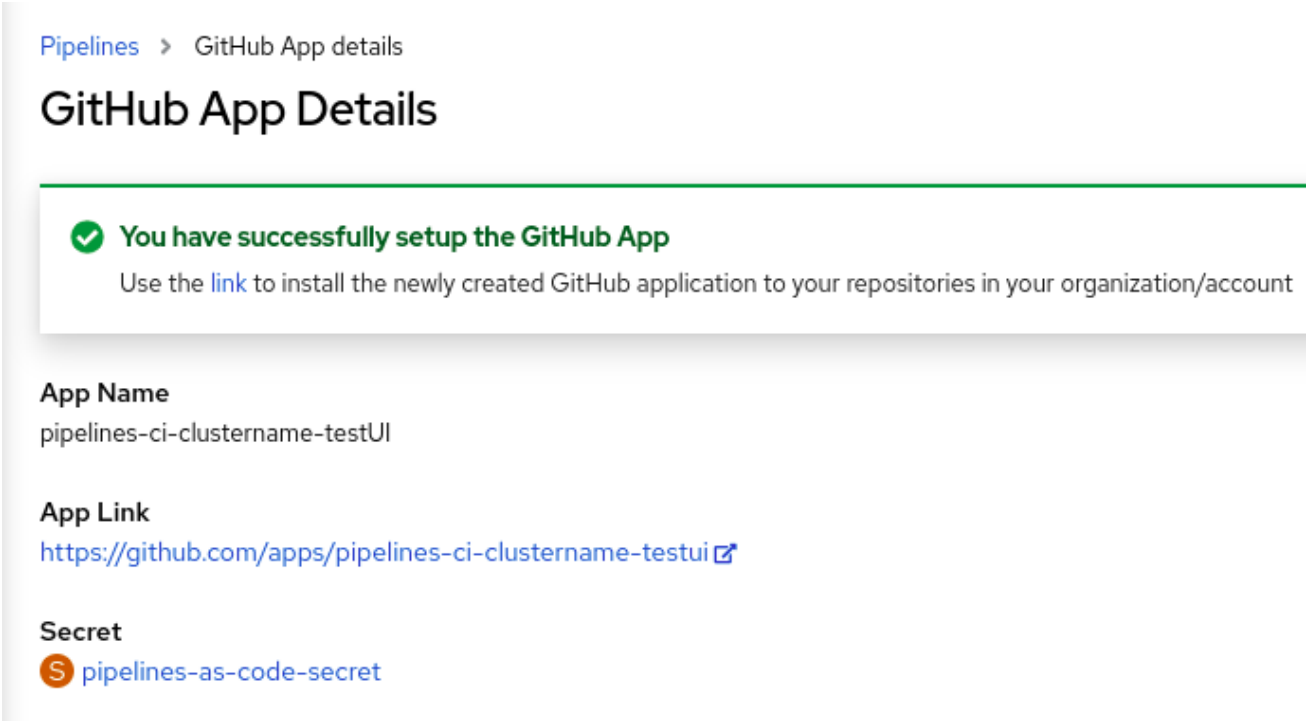
您已从 Operator Hub 安装了 Red Hat OpenShift Pipelines **pipelines-1.10** operator。

流程

1. 在 Administrator 视角中，使用导航窗格进入到 **Pipelines**。
2. 在 **Pipelines** 页面中点 **Setup GitHub App**。
3. 输入您的 GitHub 应用程序名称。例如，**pipelines-ci-clustername-testui**。
4. 点 **Setup**。
5. 在浏览器中提示时输入您的 Git 密码。
6. 点 **Create GitHub App for <username>**，其中 **<username>** 是您的 GitHub 的用户名。

验证

成功创建 GitHub 应用程序后，OpenShift Container Platform Web 控制台会打开并显示应用程序的详情。



GitHub App 的详细信息保存为 **openShift-pipelines** 命名空间中的 secret。

要查看与 GitHub 应用程序关联的名称、链接和 secret 等详情，请进入到 **Pipelines** 并点 **View GitHub App**。

4.8.6. 使用带有 GitHub Webhook 的 Pipelines as Code

如果无法创建 GitHub 应用程序，在您的仓库中使用带有 GitHub Webhook 的 Pipelines as Code。但是，使用带有 GitHub Webhook 的 Pipelines as Code 并不代表您可以访问 GitHub Check Runs API。任务的状态在拉取请求中作为注释添加，它在 **Checks** 选项卡中没有提供。

注意

带有 GitHub Webhook 的 Pipelines as Code 不支持 **/retest** 和 **/ok-to-test** 等 GitOps 注释。要重启持续集成 (CI)，请创建一个到存储库的新提交。例如，要在不进行任何更改的情况下创建新提交，您可以使用以下命令：

```
$ git --amend -a --no-edit && git push --force-with-lease <origin> <branchname>
```

先决条件

- 确保在集群中安装了 Pipelines as Code。
- 为了授权，请在 GitHub 上创建一个个人访问令牌。
  - 要生成一个安全的、细颗粒的令牌，将其范围限制为一个特定的存储库，并授予以下权限：

    表 4.7. 细粒度令牌的权限

Name	权限
管理	只读
元数据	只读
内容	只读
提交状态	读和写
Pull request	读和写
Webhook	读和写

  - 要使用经典令牌，将范围设置为 **public_repo**（公共存储库）和 **repo**（私有存储库）。另外，提供一个短的令牌过期周期，并在其他位置记录下令牌。

注意

如果要使用 **tkn pac** CLI 配置 webhook，请添加 **admin:repo_hook** 范围。

流程

1. 配置 Webhook 并创建一个 **Repository** 自定义资源 (CR)。

   - 要配置 webhook 并使用 **tkn pac** CLI 工具*自动*创建一个 **Repository** CR，请使用以下命令：

     ```
     $ tkn pac create repo
     ```

     互动输出示例

     ```
     ? Enter the Git repository url (default: https://github.com/owner/repo):
     ? Please enter the namespace where the pipeline should run (default: repo-pipelines):
     ! Namespace repo-pipelines is not found
     ? Would you like me to create the namespace repo-pipelines? Yes
     ✓ Repository owner-repo has been created in repo-pipelines namespace
     ✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
       I have detected a controller url: https://pipelines-as-code-controller-openshift-
     pipelines.apps.example.com
     ? Do you want me to use it? Yes
     ? Please enter the secret to configure the webhook for payload validation (default:
     sJNwdmTifHTs):  sJNwdmTifHTs
     ℹ  You now need to create a GitHub personal access token, please checkout the docs at
     https://docs.github.com/en/authentication/keeping-your-account-and-data-
     secure/creating-a-personal-access-token for the required scopes
     ? Please enter the GitHub access token:  ****************************************
     ✓ Webhook has been created on repository owner/repo
       Webhook Secret owner-repo has been created in the repo-pipelines namespace.
       Repository CR owner-repo has been updated with webhook secret in the repo-pipelines
     namespace
     ℹ Directory .tekton has been created.
     ✓ We have detected your repository using the programming language Go.
     ✓ A basic template has been created in
     /home/Go/src/github.com/owner/repo/.tekton/pipelinerun.yaml, feel free to customize it.
     ```

   - 要配置 webhook 并 *手动*创建 **Repository** CR，请执行以下步骤：

     i. 在 OpenShift 集群中，提取 Pipelines as Code 控制器的公共 URL。

     ```
     $ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
     jsonpath='{.spec.host}')
     ```

     ii. 在 GitHub 存储库或机构中执行以下步骤：

     A. 进入 **Settings** -> **Webhooks** 并点 **Add webhook**。

     B. 将 **Payload URL** 设置为 Pipelines as Code 的公共 URL。

     C. 将内容类型选为 **application/json**。

     D. 添加 webhook secret 并在另一个位置记录它。在本地机器上安装 **openssl** 后，生成一个随机 secret。

     ```
     $ openssl rand -hex 20
     ```

     E. 点 **Let me select individual events** 并选择这些事件：**Commit comments**, **Issue comments**, **Pull request**, 和 **Pushes**.

F. 点击 **Add webhook**。

iii. 在 OpenShift 集群中，使用个人访问令牌和 webhook secret 创建一个 **Secret** 对象。

```
$ oc -n target-namespace create secret generic github-webhook-config \
  --from-literal provider.token="<GITHUB_PERSONAL_ACCESS_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. 创建 **Repository** CR。

示例： Repository CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    secret:
      name: "github-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "github-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

注意

Pipelines as Code 假设 OpenShift **Secret** 对象和 **Repository** CR 位于同一命名空间中。

2. 可选： 对于现有的 **Repository** CR，请添加多个 GitHub Webhook secret 或为已删除的 secret 提供替换。

a. 使用 **tkn pac** CLI 工具添加 webhook。

示例：使用 tkn pac CLI 的额外 Webhook

```
$ tkn pac webhook add -n repo-pipelines
```

互动输出示例

```
✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
AeHdHTJVfAeH):  AeHdHTJVfAeH
✓ Webhook has been created on repository owner/repo
  Secret owner-repo has been updated with webhook secert in the repo-pipelines
namespace.
```

b. 更新现有 OpenShift **Secret** 对象中的 **webhook.secret** 密钥。

3. 可选：对于现有的 **Repository** CR，更新个人访问令牌。

- 使用 **tkn pac** CLI 工具更新个人访问令牌。

示例：使用 tkn pac CLI 更新个人访问令牌

```
$ tkn pac webhook update-token -n repo-pipelines
```

互动输出示例

```
? Please enter your personal access token: ****************************************
 Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
```

- 或者，通过修改 **Repository** CR 来更新个人访问令牌。

i. 在 **Repository** CR 中查找 secret 的名称。

```
...
spec:
  git_provider:
    secret:
      name: "github-webhook-config"
...
```

ii. 使用 **oc patch** 命令更新 **$target_namespace** 命名空间中的 **$NEW_TOKEN** 的值。

```
$ oc -n $target_namespace patch secret github-webhook-config -p "{\"data\": {\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

其他资源

- GitHub 中的 GitHub Webhook 文档
- GitHub 中的 GitHub Check Runs 文档
- 在 GitHub 上创建个人访问令牌
- 具有预先填充权限的经典令牌

4.8.7. 在 GitLab 中使用 Pipelines as Code

如果您的机构或项目使用 GitLab 作为首选平台，您可以在 GitLab 上使用带有 webhook 的仓库的 Pipelines as Code。

先决条件

- 确保在集群中安装了 Pipelines as Code。
- 为授权，请生成个人访问令牌作为 GitLab 上项目或机构的管理器。

注意

- 如果要使用 **tkn pac** CLI 配置 webhook，请将 **admin:repo_hook** 范围添加到令牌中。
- 使用范围被设置为针对特定项目的令牌无法提供对从已分叉存储库发送的合并请求 (MR) 的 API 访问。在这种情况下，Pipelines as Code 会以一条 MR 评论的方式显示结果。

流程

1. 配置 Webhook 并创建一个 **Repository** 自定义资源 (CR)。
   - 要配置 webhook 并使用 **tkn pac** CLI 工具*自动*创建一个 **Repository** CR，请使用以下命令：

```
$ tkn pac create repo
```

互动输出示例

```
? Enter the Git repository url (default: https://gitlab.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository repositories-project has been created in repo-pipelines namespace
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
? Please enter the project ID for the repository you want to be configured,
  project ID refers to an unique ID (e.g. 34405323) shown at the top of your GitLab project
: 17103
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
lFjHIEcaGFlF):  lFjHIEcaGFlF
ℹ  You now need to create a GitLab personal access token with `api` scope
ℹ  Go to this URL to generate one https://gitlab.com/-/profile/personal_access_tokens,
see https://is.gd/rOEo9B for documentation
? Please enter the GitLab access token:  ****************************
? Please enter your GitLab API URL::  https://gitlab.com
✓ Webhook has been created on your repository
  Webhook Secret repositories-project has been created in the repo-pipelines
namespace.
  Repository CR repositories-project has been updated with webhook secret in the repo-
pipelines namespace
ℹ Directory .tekton has been created.
✓ A basic template has been created in
/home/Go/src/gitlab.com/repositories/project/.tekton/pipelinerun.yaml, feel free to
customize it.
```

   - 要配置 webhook 并 *手动*创建 **Repository** CR，请执行以下步骤：
     i. 在 OpenShift 集群中，提取 Pipelines as Code 控制器的公共 URL。

```
$ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
jsonpath='{.spec.host}')
```

ii. 在 GitLab 项目中，执行以下步骤：

A. 使用左侧边栏进入 **Settings** -> **Webhooks**。

B. 将 **URL** 设置为 Pipelines as Code 控制器公共 URL。

C. 添加 webhook secret 并在另一个位置记录它。在本地机器上安装 **openssl** 后，生成一个随机 secret。

```
$ openssl rand -hex 20
```

D. 点 **Let me select individual events** 并选择这些事件：**Commit comments**, **Issue comments**, **Pull request**, 和 **Pushes**.

E. 点 **Save Changes**。

iii. 在 OpenShift 集群中，使用个人访问令牌和 webhook secret 创建一个 **Secret** 对象。

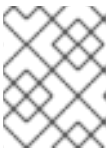
```
$ oc -n target-namespace create secret generic gitlab-webhook-config \
  --from-literal provider.token="<GITLAB_PERSONAL_ACCESS_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. 创建 **Repository** CR。

示例： Repository CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://gitlab.com/owner/repo" 1
  git_provider:
    secret:
      name: "gitlab-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "gitlab-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

1 目前，Pipelines as Code 不会自动检测 GitLab 的私有实例。在这种情况下，在 **git_provider.url** spec 下指定 API URL。通常，您可以使用 **git_provider.url** spec 手动覆盖 API URL。



注意

- Pipelines as Code 假设 OpenShift **Secret** 对象和 **Repository** CR 位于同一命名空间中。

2. 可选： 对于现有的 **Repository** CR，请添加多个 GitLab Webhook secret 或为已删除的 secret 提供替换。

a. 使用 **tkn pac** CLI 工具添加 webhook。

示例：使用 tkn pac CLI 添加额外的 Webhook

```
$ tkn pac webhook add -n repo-pipelines
```

互动输出示例

```
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
 I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default: AeHdHTJVfAeH):  AeHdHTJVfAeH
✓ Webhook has been created on repository owner/repo
 Secret owner-repo has been updated with webhook secert in the repo-pipelines namespace.
```

b. 更新现有 OpenShift **Secret** 对象中的 **webhook.secret** 密钥。

3. 可选：对于现有的 **Repository** CR，更新个人访问令牌。

- 使用 **tkn pac** CLI 工具更新个人访问令牌。

示例：使用 tkn pac CLI 更新个人访问令牌

```
$ tkn pac webhook update-token -n repo-pipelines
```

互动输出示例

```
? Please enter your personal access token:  ****************************************
 Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
```

- 或者，通过修改 **Repository** CR 来更新个人访问令牌。

i. 在 **Repository** CR 中查找 secret 的名称。

```
...
spec:
  git_provider:
    secret:
      name: "gitlab-webhook-config"
...
```

ii. 使用 **oc patch** 命令更新 **$target_namespace** 命名空间中的 **$NEW_TOKEN** 的值。

```
$ oc -n $target_namespace patch secret gitlab-webhook-config -p "{\"data\": {\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

其他资源

- GitLab 上的 GitLab Webhook 文档

4.8.8. 在 Bitbucket Cloud 中使用 Pipelines as Code

如果您的机构或项目使用 Bitbucket Cloud 作为首选平台，您可以在 Bitbucket Cloudb 上使用带有 webhook 的仓库的 Pipelines as Code。

先决条件

- 确保在集群中安装了 Pipelines as Code。
- 在 Bitbucket Cloud 上创建一个应用程序密码。
  - 选中以下框，为令牌添加适当的权限：
    - 账户：**Email**, **Read**
    - 工作区成员资格： **Read**、**Write**
    - 项目： **Read**,**Write**
    - Issues： **Read**,**Write**
    - Pull requests: **Read**,**Write**

注意

- 如果要使用 **tkn pac** CLI 配置 webhook，请在令牌中添加 **Webhooks**:**Read** 和 **Write** 权限。
- 生成后，在另外一个位置保持密码或令牌的副本。

流程

1. 配置 Webhook 并创建一个 **Repository** CR。
   - 要配置 webhook 并使用 **tkn pac** CLI 工具*自动*创建一个 **Repository** CR，请使用以下命令：

```
$ tkn pac create repo
```

互动输出示例

```
? Enter the Git repository url (default: https://bitbucket.org/workspace/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository workspace-repo has been created in repo-pipelines namespace
✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
? Please enter your bitbucket cloud username:  <username>
ℹ  You now need to create a Bitbucket Cloud app password, please checkout the docs at https://is.gd/fqMHiJ for the required permissions
? Please enter the Bitbucket Cloud app password:  ************************************
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
✓ Webhook has been created on repository workspace/repo
  Webhook Secret workspace-repo has been created in the repo-pipelines namespace.
  Repository CR workspace-repo has been updated with webhook secret in the repo-
```

```
pipelines namespace
ℹ Directory .tekton has been created.
✓ A basic template has been created in
/home/Go/src/bitbucket/repo/.tekton/pipelinerun.yaml, feel free to customize it.
```

- 要配置 webhook 并 *手动*创建 **Repository** CR，请执行以下步骤：

  i. 在 OpenShift 集群中，提取 Pipelines as Code 控制器的公共 URL。

  ```
  $ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
  jsonpath='{.spec.host}')
  ```

  ii. 在 Bitbucket Cloud 上，执行以下步骤：

  A. 使用 Bitbucket Cloud 存储库的左侧导航窗格，进入 **Repository settings** -> **Webhooks**，然后点 **Add webhook**。

  B. 设置 **Title**。例如，"Pipelines as Code"。

  C. 将 **URL** 设置为 Pipelines as Code 控制器公共 URL。

  D. 选择这些事件： **Repository: Push**,**Pull Request: Created**,**Pull Request: Updated**, 和 **Pull Request: Comment created**.

  E. 点击 **Save**。

  iii. 在 OpenShift 集群中，使用目标命名空间中的 app 密码创建一个 **Secret** 对象。

  ```
  $ oc -n target-namespace create secret generic bitbucket-cloud-token \
    --from-literal provider.token="<BITBUCKET_APP_PASSWORD>"
  ```

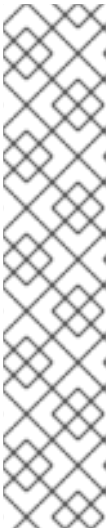
  iv. 创建 **Repository** CR。

  示例： Repository CR

  ```
  apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
  kind: Repository
  metadata:
    name: my-repo
    namespace: target-namespace
  spec:
    url: "https://bitbucket.com/workspace/repo"
    branch: "main"
    git_provider:
      user: "<BITBUCKET_USERNAME>" 1
      secret:
        name: "bitbucket-cloud-token" 2
        key: "provider.token" # Set this if you have a different key in your secret
  ```

  1 您只能通过所有者文件中的 **ACCOUNT_ID** 引用用户。

  2 Pipelines as Code 假设 **git_provider.secret** spec 引用的 secret 和 **Repository** CR 位于同一命名空间中。



注意

- Bitbucket 云不支持 **tkn pac create** 和 **tkn pac bootstrap** 命令。
- Bitbucket 云不支持 Webhook secret。为了保护有效负载并防止 CI 被劫持，Pipelines as Code 会获取 Bitbucket 云 IP 地址列表，并确保 Webhook 接收仅来自这些 IP 地址。
  - 要禁用这个默认行为，在 Pipelines 中将 **bitbucket-cloud-check-source-ip** 键设置为 **false**，作为 **pipelines-as-code** 命名空间的代码配置映射中。
  - 要允许额外的安全 IP 地址或网络，请将它们作为逗号分隔的值添加到 Pipelines 中的 **bitbucket-cloud-additional-source-ip** 键中，作为 **pipelines-as-code** 命名空间的代码配置映射。

2. 可选：对于现有的 **Repository** CR，请添加多个 Bitbucket Cloud Webhook secret 或为已删除的 secret 提供替换。

   a. 使用 **tkn pac** CLI 工具添加 webhook。

   示例：使用 tkn pac CLI 添加额外的 Webhook

   ```
   $ tkn pac webhook add -n repo-pipelines
   ```

   互动输出示例

   ```
   ✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
   ? Please enter your bitbucket cloud username:  <username>
     I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
   ? Do you want me to use it? Yes
   ✓ Webhook has been created on repository workspace/repo
     Secret workspace-repo has been updated with webhook secret in the repo-pipelines namespace.
   ```

   注意

   只有在 **Repository** CR 在 default 命名空间以外的命名空间中存在时，才在运行 **tkn pac webhook add** 命令时使用 **[-n <namespace>]** 选项。

   b. 更新现有 OpenShift **Secret** 对象中的 **webhook.secret** 密钥。

3. 可选：对于现有的 **Repository** CR，更新个人访问令牌。

   - 使用 **tkn pac** CLI 工具更新个人访问令牌。

   示例：使用 tkn pac CLI 更新个人访问令牌

   ```
   $ tkn pac webhook update-token -n repo-pipelines
   ```

   互动输出示例

```
? Please enter your personal access token: ****************************************
  Secret owner-repo has been updated with new personal access token in the repo-
pipelines namespace.
```

注意

只有在 **Repository** CR 在 default 命名空间以外的命名空间中存在时，才在 **tkn pac webhook update-token** 命令中使用 **[-n <namespace>]** 选项。

- 或者，通过修改 **Repository** CR 来更新个人访问令牌。

  i. 在 **Repository** CR 中查找 secret 的名称。

```
...
spec:
  git_provider:
    user: "<BITBUCKET_USERNAME>"
    secret:
      name: "bitbucket-cloud-token"
      key: "provider.token"
...
```

  ii. 使用 **oc patch** 命令更新 **$target_namespace** 命名空间中的 **$password** 的值。

```
$ oc -n $target_namespace patch secret bitbucket-cloud-token -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

其他资源

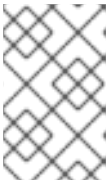
- 在 Bitbucket 云上创建应用程序密码
- Altassian 帐户 ID 和 Nicknames

4.8.9. 在 Bitbucket Server 中使用 Pipelines as Code

如果您的机构或项目使用 Bitbucket Cloud 作为首选平台，您可以在 Bitbucket Server 上使用带有 webhook 的仓库的 Pipelines as Code。

先决条件

- 确保在集群中安装了 Pipelines as Code。
- 在 Bitbucket 服务器上，生成个人访问令牌作为项目管理器，并将它保存到一个其他位置中。



注意

- 令牌必须具有 **PROJECT_ADMIN** 和 **REPOSITORY_ADMIN** 权限。
- 令牌必须有权访问拉取请求中分叉的存储库。

流程

1. 在 OpenShift 集群中，提取 Pipelines as Code 控制器的公共 URL。

```
$ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
jsonpath='{.spec.host}')
```

2. 在 Bitbucket 服务器上，执行以下步骤：

   a. 使用 Bitbucket Data Center 存储库的左侧导航窗格，进入 **Repository settings** -> **Webhooks** 并点 **Add webhook**。

   b. 设置 **Title**。例如，"Pipelines as Code"。

   c. 将 **URL** 设置为 Pipelines as Code 控制器公共 URL。

   d. 添加 webhook secret，并将它的副本保存到备用位置。如果您在本地机器上安装了 **openssl**，使用以下命令生成随机 secret：

      ```
      $ openssl rand -hex 20
      ```

   e. 选择以下事件：

      - **Repository: Push**
      - **Repository: Modified**
      - **Pull Request: Opened**
      - **Pull Request: Source branch updated**
      - **Pull Request: Comment added**

   f. 点击 **Save**。

3. 在 OpenShift 集群中，使用目标命名空间中的 app 密码创建一个 **Secret** 对象。

```
$ oc -n target-namespace create secret generic bitbucket-server-webhook-config \
  --from-literal provider.token="<PERSONAL_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

4. 创建 **Repository** CR。

示例： Repository CR

```
---
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  git_provider:
    url: "https://bitbucket.server.api.url/rest" (1)
    user: "<BITBUCKET_USERNAME>" (2)
    secret: (3)
      name: "bitbucket-server-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
```

```
  webhook_secret:
    name: "bitbucket-server-webhook-config"
    key: "webhook.secret" # Set this if you have a different key for your secret
```

1. 确保具有不带 **/api/v1.0** 后缀的正确的 Bitbucket 服务器 API URL。通常，默认安装有一个 **/rest** 后缀。
2. 您只能通过所有者文件中的 **ACCOUNT_ID** 引用用户。
3. Pipelines as Code 假设 **git_provider.secret** spec 引用的 secret 和 **Repository** CR 位于同一命名空间中。

注意

Bitbucket 服务器不支持 **tkn pac create** 和 **tkn pac bootstrap** 命令。

其他资源

- 在 Bitbucket 服务器上创建个人令牌
- 在 Bitbucket 服务器上创建 Webhook

4.8.10. 使用自定义证书与 Pipelines as Code 进行交互

要将 Pipelines as Code 配置为使用通过私有签名或自定义证书进行访问的 Git 存储库，您可以将证书公开给 Pipelines as Code。

流程

- 如果您使用 Red Hat OpenShift Pipelines Operator 安装了 Pipelines as Code，则可以使用 **Proxy** 对象将自定义证书添加到集群中。Operator 在所有 Red Hat OpenShift Pipelines 组件和工作负载中公开证书，包括 Pipelines as Code。

其他资源

- 启用集群范围代理

4.8.11. 在 Pipelines as Code 中使用 Repository CRD

Repository 自定义资源 (CR) 有以下主要功能：

- 告知 Pipelines as Code 关于处理来自一个 URL 事件的信息。
- 告知 Pipelines as Code 关于管道运行的命名空间信息。
- 使用 webhook 方法时，对于 Git 供应商平台需要需要引用 API secret、用户名或 API URL。
- 为存储库提供最后的管道运行状态。

您可以使用 **tkn pac** CLI 或其他替代方法在目标命名空间中创建 **Repository** CR。例如：

```
cat <<EOF|kubectl create -n my-pipeline-ci -f- 1
```

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: project-repository
spec:
  url: "https://github.com/<repository>/<project>"
EOF
```

1 **my-pipeline-ci** 是目标命名空间。

每当有来自 URL（如 **https://github.com/<repository>/<project>**）的事件，Pipelines as Code 会匹配它并开始为管道运行签出 **<repository>/<project>** 存储库的内容以匹配 **.tekton/** 目录中的内容。

注意

- 您必须在与要执行的源代码存储库关联的管道所在的同一命名空间中创建 **Repository** CRD，它不能针对不同的命名空间。
- 如果多个 **Repository** CRD 与同一事件匹配，Pipelines as Code 将只会处理最老的一个。如果您需要与特定命名空间匹配，请添加 **pipelinesascode.tekton.dev/target-namespace: "<mynamespace>"** 注解。通过这种显式目标，可防止恶意攻击者在其无法访问的命名空间中执行一个管道运行。

4.8.11.1. 在 Repository CRD 中设置并发限制

您可以使用 **Repository** CRD 中的 **concurrency_limit** spec 来定义为存储库同时运行的最大管道运行数。

```
...
spec:
  concurrency_limit: <number>
  ...
```

如果有多个管道与事件匹配，管道会按照字母顺序与事件启动匹配。

例如，如果您在 **.tekton** 目录中有三个管道运行，并在存储库配置中创建了一个 **concurrency_limit** 为 **1** 的拉取请求，则所有管道运行都会按字母顺序执行。在任意给定时间点，只有一个管道运行处于 running 状态，而其余会处于排队状态。

4.8.12. 使用 Pipelines as Code 解析器

Pipelines as Code 解析器可确保运行的管道运行不会与其他其它运行冲突。

要分割您的管道和管道运行，请将文件存储在 **.tekton/** 目录或其子目录中。

如果 Pipelines as Code 在 **.tekton/** 目录中的任何 YAML 文件中发现一个带有对任务或管道的引用的管道运行，Pipelines as Code 会自动解析引用的任务，以使用 **PipelineRun** 对象中嵌入的 spec 提供单个管道运行。

如果 Pipelines as Code 无法解析 **Pipeline** 或 **PipelineSpec** 定义中引用的任务，则运行会在对集群应用更改前失败。您可以在 Git 供应商平台上查看问题，在 **Repository** CR 所在的目标命名空间的事件内。

当解析器观察到以下类型的任务时会跳过解析过程：

- 对一个集群任务的引用。
- 任务或管道捆绑包。
- 具有没有 **tekton.dev/** 前缀的 API 版本的自定义任务。

解析器以字面形式使用此类任务，不进行任何转换。

如果要在拉取请求中发送它之前在本地对管道运行进行测试，请使用 **tkn pac resolve** 命令。

您还可以引用远程管道和任务。

4.8.12.1. 在 with Pipelines as Code 中使用远程任务注解

Pipelines as Code 支持在一个管道运行中使用注解来获取远程任务或管道。如果您在管道运行或 **PipelineRun** 或 **PipelineSpec** 对象中引用远程任务，Pipelines as Code 码解析器会自动包含它。如果在获取远程任务或解析它们时出现错误，Pipelines as Code 将停止处理任务。

要包含远程任务，请参阅以下注解示例：

引用 Tekton Hub 中的远程任务

- 在 Tekton Hub 中引用单个远程任务。

```
...
  pipelinesascode.tekton.dev/task: "git-clone" (1)
...
```

(1) Pipelines as Code 包括来自 Tekton Hub 的任务的最新版本。

- 引用 Tekton Hub 中的多个远程任务

```
...
  pipelinesascode.tekton.dev/task: "[git-clone, golang-test, tkn]"
...
```

- 使用 **-<NUMBER>** 后缀引用 Tekton Hub 中的多个远程任务。

```
...
  pipelinesascode.tekton.dev/task: "git-clone"
  pipelinesascode.tekton.dev/task-1: "golang-test"
  pipelinesascode.tekton.dev/task-2: "tkn" (1)
...
```

(1) 默认情况下，Pipelines as Code 会将字符串解析为从 Tekton Hub 获取最新的任务。

- 引用 Tekton Hub 中的远程任务的特定版本。

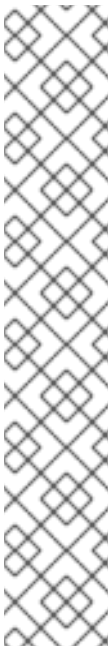
```
...
  pipelinesascode.tekton.dev/task: "[git-clone:0.1]" (1)
...
```

1 引用 Tekton Hub 中的 **git-clone** 远程任务的 **0.1** 版本。

使用 URL 的远程任务

```
...
  pipelinesascode.tekton.dev/task: "<https://remote.url/task.yaml>" 1
...
```

1 远程任务的公共 URL。



注意

- 如果使用 GitHub，远程任务 URL 使用与 **Repository** CRD 相同的主机，Pipelines as Code 会使用 GitHub 令牌并使用 GitHub API 获取 URL。例如，您有一个类似 **https://github.com/<organization>/<repository>** 的存储库 URL，远程 HTTP URL 引用一个与 **https://github.com/<organization>/<repository>/blob/<mainbranch>/<path>/<file>** 类似的 GitHub blob，Pipelines as Code 会使用 GitHub App 令牌从私有存储库中抓取任务定义。

  当您使用公共 GitHub 存储库时，Pipelines as Code 的行为与 GitHub 原始 URL（如 **https://raw.githubusercontent.com/<organization>/<repository>/<main branch>/<path>/<file>**）类似。

- GitHub App 令牌的范围为所有者或存储库所在的机构。使用 GitHub Webhook 方法时，您可以在允许个人令牌的任何机构中获取任何私有或公共存储库。

从存储库内的 YAML 文件中引用任务

```
...
pipelinesascode.tekton.dev/task: "<share/tasks/git-clone.yaml>" 1
...
```

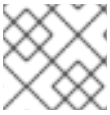
1 到包含任务定义的本地文件的相对路径。

4.8.12.2. 在 Pipelines as Code 中使用远程管道注解

您可以使用远程管道注解在多个存储库间共享管道定义。

```
...
    pipelinesascode.tekton.dev/pipeline: "<https://git.provider/raw/pipeline.yaml>" 1
...
```

1 远程管道定义的 URL。您还可以为同一存储库中的文件提供位置。



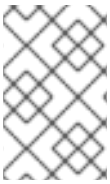
注意

您只能使用注解引用一个管道定义。

4.8.13. 使用 Pipelines as Code 创建管道运行

要使用 Pipelines as Code 运行管道，您可以在存储库的 **.tekton/** 目录中创建管道定义或模板作为 YAML 文件。您可以使用远程 URL 在其他存储库中引用 YAML 文件，但管道运行仅由包含 **.tekton/** 目录中的事件触发。

Pipelines as Code 解析器会将所有任务与管道运行绑定，作为一个单独的管道运行，而无需外部依赖项。



注意

- 对于管道，使用最少一个带有 spec 的管道，或一个独立的 **Pipeline** 对象。
- 对于任务，在管道中嵌入任务 spec，或者将其单独定义为一个 Task 对象。

参数化提交和 URL

您可以使用 {{<var>}} 的格式来使用动态的、可扩展的参数，在您的提交中指定参数。目前，您可以使用以下变量：

- **{{repo_owner}}**: 存储库所有者。
- **{{repo_name}}**: 存储库名称。
- **{{repo_url}}**：存储库完整 URL。
- **{{revision}}**：提交的完全 SHA 修订。
- **{{sender}}**：提交发送者的用户名或帐户 ID。
- **{{source_branch}}**：事件源自的分支名称。
- **{{target_branch}}**：事件目标的分支名称。对于推送事件，它与 **source_branch** 相同。
- **{{pull_request_number}}**：拉取或合并请求号，仅对 **pull_request** 事件类型定义。
- **{{git_auth_secret}}**：使用 Git 提供程序的令牌自动生成的 secret 名称，用于签出私有存储库。

将事件与管道运行匹配

您可以通过在管道运行中使用特殊注解，将不同的 Git 供应商事件与每个管道匹配。如果有多个管道与事件匹配，Pipelines as Code 会并行运行它们，并在管道运行完成后将结果发送到 Git 供应商。

将 pull 事件与管道运行匹配

您可以使用以下示例将 **pipeline-pr-main** 管道与以 **main** 分支为目标的 **pull_request** 事件匹配：

```
...
  metadata:
    name: pipeline-pr-main
  annotations:
    pipelinesascode.tekton.dev/on-target-branch: "[main]" 1
    pipelinesascode.tekton.dev/on-event: "[pull_request]"
...
```

-

1 您可以通过添加以逗号分隔的条目来指定多个分支。例如："**[main, release-nightly]**"。另外，您可以指定以下内容：

- 对分支的完整引用，如 **"refs/heads/main"**
- 带有模式匹配的 globs，如 **"refs/heads/* the"**
- 标签，如 **"refs/tags/1.*"**

将推送事件与管道运行匹配

您可以使用以下示例将 **pipeline-push-on-main** 管道与以 **refs/heads/main** 分支为目标的 **push** 事件匹配：

```
...
  metadata:
    name: pipeline-push-on-main
  annotations:
    pipelinesascode.tekton.dev/on-target-branch: "[refs/heads/main]" 1
    pipelinesascode.tekton.dev/on-event: "[push]"
...
```

1 您可以通过添加以逗号分隔的条目来指定多个分支。例如："**[main, release-nightly]**"。另外，您可以指定以下内容：

- 对分支的完整引用，如 **"refs/heads/main"**
- 带有模式匹配的 globs，如 **"refs/heads/* the"**
- 标签，如 **"refs/tags/1.*"**

高级事件匹配

Pipelines as Code 支持在高级事件匹配中使用基于通用表达式语言 (CEL) 的过滤。如果您在管道运行中有 **pipelinesascode.tekton.dev/on-cel-expression** 注解，PPipelines as Code 使用 CEL 表达式并跳过 **on-target-branch** 注解。与简单的 **on-target-branch** 注解匹配相比，CEL 表达式允许复杂的过滤和负效果。

要将基于 CEL 的过滤作为代码使用，请考虑以下注解示例：

- 匹配以 **main** 分支为目标并来自 **wip** 分支的 **pull_request** 事件：

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch == "main" && source_branch == "wip"
...
```

- 要仅在路径更改时运行管道，您可以使用带有 glob 模式的 **.pathChanged** 后缀功能：

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && "docs/*.md".pathChanged() 1
...
```

1 匹配 **docs** 目录中的所有 markdown 文件。

- 匹配以标题 **[DOWNSTREAM]** 开头的所有拉取请求：

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request && event_title.startsWith("[DOWNSTREAM]")
...
```

- 要在 **pull_request** 事件上运行管道，但跳过 **experimental** 分支：

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch != experimental"
...
```

对于使用 Pipelines as Code 时进行基于 CEL 的高级过滤，您可以使用以下字段和后缀功能：

- **event**: 一个 **push** 或 **pull_request** 事件。
- **target_branch**：目标分支。
- **source_branch**: 一个 **pull_request** 事件的源分支。对于 **push** 事件，它与 **target_branch** 相同。
- **event_title** ：匹配事件的标题，如一个 **push** 事件的提交标题，以及 **pull_request** 事件的 pull 或 merge 请求的标题。目前，只有 GitHub、Gitlab 和 Bitbucket 云是受支持的提供程序。
- **.pathChanged**：字符串的后缀函数。如果路径已改变，字符串可以是一个路径的 glob。目前，只支持 GitHub 和 Gitlab。

使用临时 GitHub App 令牌用于 Github API 操作

您可以使用来自 GitHub App 的 Pipelines as Code 生成的临时安装令牌来访问 GitHub API。令牌值存储在为私有存储库生成的临时 **{{git_auth_secret}}** 动态变量的 **git-provider-token** 键中。

例如，要在拉取请求中添加注释，您可以使用 Tekton Hub 中的 **github-add-comment** 任务使用 Pipelines as Code 注解：

```
...
  pipelinesascode.tekton.dev/task: "github-add-comment"
...
```

然后，您可以在 **tasks** 部分添加任务，或管道运行定义中的 **finally** 任务：

```
[...]
tasks:
  - name:
      taskRef:
        name: github-add-comment
      params:
        - name: REQUEST_URL
          value: "{{ repo_url }}/pull/{{ pull_request_number }}" 1
        - name: COMMENT_OR_FILE
```

```
      value: "Pipelines as Code IS GREAT!"
    - name: GITHUB_TOKEN_SECRET_NAME
      value: "{{ git_auth_secret }}"
    - name: GITHUB_TOKEN_SECRET_KEY
      value: "git-provider-token"
...
```

1 通过使用动态变量，您可以为来自任何存储库的任何拉取请求重复使用此片断模板。

注意

在 GitHub 应用程序中，生成的安装令牌的有效期为 8 小时，其范围限制于事件源自于集群中配置的存储库。

其他资源

- CEL 语言规格

4.8.14. 使用 Pipelines as Code 运行管道运行

使用默认配置，当指定事件（如拉取请求或推送在仓库上发生）时，Pipelines as Code 会在默认分支的 **.tekton/** 目录中运行任何管道运行。例如，如果在默认分支上运行的管道具有注解 **pipelinesascode.tekton.dev/on-event: "[pull_request]"**，它将在每次发生拉取请求事件时运行。

如果是拉取请求或合并请求，Pipelines as Code 也会从默认分支以外的分支运行管道，如果拉取请求作者满足以下条件：

- 作者是存储库的所有者。
- 作者是存储库的合作者。
- 作者是存储库机构中的公共成员。
- 拉取请求作者列在位于 **main** 分支存储库根的 **OWNER** 文件中，如存储库的 GitHub 配置中定义的。另外，拉取请求作者被添加到 **approvers** 或 **reviewers** 部分。例如，如果作者在 **approvers** 部分中列出，则作者引发的拉取请求将启动管道运行。

```
...
  approvers:
    - approved
...
```

如果拉取请求作者没有满足要求，则满足要求的其他用户可以在拉取请求中注释掉 **/ok-to-test**，并启动管道运行。

Pipeline 运行执行

管道运行始终在与生成事件的存储库关联的 **Repository** CRD 的命名空间中运行。

您可以使用 **tkn pac** CLI 工具观察管道运行的执行。

- 要遵循最后一次管道运行的执行，请使用以下示例：

```
$ tkn pac logs -n <my-pipeline-ci> -L 1
```

1 **my-pipeline-ci** 是 **Repository** CRD 的命名空间。

- 要以交互方式执行任何管道运行，请使用以下示例：

```
$ tkn pac logs -n <my-pipeline-ci> 1
```

1 **my-pipeline-ci** 是 **Repository** CRD 的命名空间。如果需要查看除最后一个管道运行以外的管道运行，您可以使用 **tkn pac logs** 命令选择附加到仓库的 **PipelineRun** ：

如果您配置使用 GitHub App 的 Pipelines as Code，Pipelines as Code 会在 GitHub App 的 **Checks** 选项卡中发布了一个 URL。您可以点 URL 并按照管道执行进行操作。

重启管道运行

您可以重启没有事件的管道运行，如向分支发送新提交或提升拉取请求。在 GitHub App 上，进入 **Checks** 选项卡，然后点 **Re-run**。

如果您以 pull 或 merge 请求为目标，请在拉取请求中使用以下注释来重启所有或特定的管道运行：

- **/retest** 注释会重启所有管道运行。
- **/retest <pipelinerun-name>** 注释重启特定的管道运行。
- **/cancel** 注释取消所有管道运行。
- **/cancel <pipelinerun-name>** 注释会取消特定的管道运行。

注释的结果在 GitHub App 的 **Checks** 选项卡中显示。

4.8.15. 使用 Pipelines as Code 监控管道运行状态

根据上下文和支持的工具，您可以以不同的方式监控管道运行的状态。

GitHub 应用程序的状态

当管道运行完成后，在 **Check** 选项卡中添加状态，其中包含管道的每个任务以及 **tkn pipelinerun describe** 命令的输出的有限信息。

日志错误片断

当 Pipelines as Code 检测到管道其中一个任务中的错误时，会显示一段信息，它包括第一个失败的任务的任务详情中的最后 3 行信息。

注意

Pipelines as Code 通过查看管道运行并使用隐藏字符替换 secret 值来避免泄漏 secret。但是，Pipelines as Code 无法隐藏来自工作区和 envFrom 源的 secret。

日志错误片断的注解

在 Pipelines as Code 配置映射中，如果您将 **error-detection-from-container-logs** 参数设置为 **true**，Pipelines as Code 会检测容器日志中的错误，并在发生错误的拉取请求中添加它们作为注解。

重要

这个功能只是一个技术预览。

目前，Pipelines as Code 只支持使用以下格式的 **makefile** 或 **grep** 输出的简单情况：

```
<filename>:<line>:<column>: <error message>
```

您可以使用 **error-detection-simple-regexp** 字段自定义用于检测错误的正则表达式。正则表达式使用命名组来授予如何指定匹配的灵活性。需要匹配的组包括文件名、行和错误。您可以查看 Pipelines as Code 配置映射用于默认正则表达式。

注意

默认情况下，Pipelines as Code 只扫描容器日志的最后 50 行。您可以在 **error-detection-max-number-of-lines** 字段中增加这个值，或为无限数量的行设置 **-1**。但是，此类配置可能会增加监视器的内存用量。

Webhook 的状态

对于 Webhook，当事件是拉取请求时，状态将添加为拉取请求或合并请求的注释。

失败

如果命名空间与 **Repository** CRD 匹配，Pipelines as Code 会在命名空间中的 Kubernetes 事件中发出其失败日志消息。

与 Repository CRD 关联的状态

管道运行的最后 5 个状态消息存储在 **Repository** 自定义资源中。

```
$ oc get repo -n <pipelines-as-code-ci>
```

```
NAME                 URL                                                NAMESPACE            SUCCEEDED
REASON      STARTTIME   COMPLETIONTIME
pipelines-as-code-ci   https://github.com/openshift-pipelines/pipelines-as-code   pipelines-as-code-ci
True        Succeeded   59m         56m
```

使用 **tkn pac describe** 命令，您可以提取与存储库及其元数据关联的运行状态。

通知

Pipelines as Code 不管理通知。如果您需要通知，请使用管道的 **finally** 功能。

其他资源

- 成功或失败时发送 Slack 消息的示例任务
- 一个管道运行示例，带有 **finally** 任务在推送事件中触发

4.8.16. 在 Pipelines as Code 中使用私有存储库

Pipelines as Code 通过使用用户令牌在目标命名空间中创建或更新 secret 来支持私有存储库。Tekton Hub 中的 **git-clone** 任务使用用户令牌来克隆私有存储库。

每当作为代码在目标命名空间中运行时，它会使用 **pac-gitauth-<REPOSITORY_OWNER>-<REPOSITORY_NAME>-<RANDOM_STRING>** 格式创建或更新 secret。

您必须使用管道运行和管道定义中的 **basic-auth** 工作区来引用 secret，然后传递给 **git-clone** 任务。

```
...
  workspace:
  - name: basic-auth
    secret:
      secretName: "{{ git_auth_secret }}"
...
```

在管道中，您可以引用 **git-clone** 任务的 **basic-auth** 工作区来重复使用：

```
...
workspaces:
  - name basic-auth
params:
    - name: repo_url
    - name: revision
...
tasks:
  workspaces:
    - name: basic-auth
      workspace: basic-auth
  ...
  tasks:
  - name: git-clone-from-catalog
      taskRef:
        name: git-clone (1)
      params:
        - name: url
          value: $(params.repo_url)
        - name: revision
          value: $(params.revision)
...
```

(1) **git-clone** 任务获取 **basic-auth** 工作区，并使用它来克隆私有存储库。

根据 Pipelines as Code 配置映射中的要求，您可以通过将 **secret-auto-create** 标志设置为 **false** 或 **true** 值来修改此配置。

其他资源

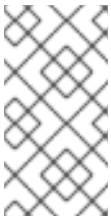
- 一个 **git-clone** 任务实例，用于克隆私有存储库

4.8.17. 使用 Pipelines as Code 清理管道运行

在一个用户命名空间中可以运行多个管道。通过设置 **max-keep-runs** 注解，您可以将 Pipelines as Code 配置为保留与事件匹配的有限数量的管道运行。例如：

```
...
  pipelinesascode.tekton.dev/max-keep-runs: "<max_number>" 1
...
```

1 Pipelines as Code 在成功完成执行后启动清理，只保留使用注解配置的最大管道运行数量。



注意

- Pipelines as Code 会跳过清理运行的管道，但会清理带有未知状态的管道运行。
- Pipelines as Code 会跳过清理失败的拉取请求。

4.8.18. 在 Pipelines as Code 中使用传入（incoming）webhook

使用传入的 webhook URL 和共享 secret，您可以在存储库中启动管道运行。

要使用传入的 Webhook，请在 **Repository** CRD 的 **spec** 部分指定以下内容：

- Pipelines as Code 匹配的传入 Webhook URL。
- Git 提供程序和用户令牌。目前，Pipelines as Code 支持 **github**、**gitlab** 和 **bitbucket-cloud**。

注意

在 GitHub 应用程序上下文中使用传入的 Webhook URL 时，您必须指定令牌。

- 目标分支和传入 Webhook URL 的 secret。

示例：带有传入 Webhook 的 Repository CRD

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: repo
  namespace: ns
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    type: github
    secret:
      name: "owner-token"
  incoming:
    - targets:
      - main
      secret:
        name: repo-incoming-secret
      type: webhook-url
```

示例：传入 webhook 的 repo-incoming-secret secret

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: repo-incoming-secret
  namespace: ns
type: Opaque
stringData:
  secret: <very-secure-shared-secret>
```

要触发位于 Git 存储库的 **.tekton** 目录中的管道运行，请使用以下命令：

```
$ curl -X POST 'https://control.pac.url/incoming?secret=very-secure-shared-
secret&repository=repo&branch=main&pipelinerun=target_pipelinerun'
```

Pipelines as Code 与传入的 URL 匹配，并将其视为 **push** 事件。但是，Pipelines as Code 不会报告这个命令触发的管道运行的状态。

要获取报告或通知，请将 **finally** 任务直接添加到您的管道中。另外，您可以使用 **tkn pac** CLI 工具检查 **Repository** CRD。

4.8.19. 自定义 Pipelines as Code 配置

要自定义 Pipelines as Code，集群管理员可使用 **pipelines-as-code** 命名空间中的 **pipelines-as-code** 配置映射配置以下参数：

表 4.8. 自定义 Pipelines as Code 配置

参数	描述	default
application-name	应用程序的名称。例如，GitHub Checks 标签中显示的名称。	**"Pipelines as Code CI"**
max-keep-days	执行的管道运行的天数保存在 **pipelines-as-code 命名空间**中。 请注意，此配置映射设置不会影响用户的管道运行的清理，这由用户 GitHub 仓库中管道运行定义上的注解控制。	
secret-auto-create	指明是否应使用 GitHub 应用中生成的令牌自动创建 secret。然后可将这个 secret 用于私有仓库。	**enabled**
remote-tasks	启用后，允许来自管道运行注解的远程任务。	**enabled**
hub-url	Tekton Hub API 的基本 URL。	**https://hub.tekton.dev/**
hub-catalog-name	Tekton Hub 目录名称。	**tekton**

参数	描述	default
tekton-dashboard-url	Tekton Hub 仪表板的 URL。Pipelines as Code 使用这个 URL 在 Tekton Hub 仪表板中生成一个 **PipelineRun** URL。	不适用
bitbucket-cloud-check-source-ip	通过查询公共 Bitbucket 的 IP 范围来指示是否保护服务请求。更改参数的默认值可能会导致安全问题。	**enabled**
bitbucket-cloud-additional-source-ip	指明是否提供一组额外的 IP 范围或网络，它们用逗号分开。	不适用
max-keep-run-upper-limit	管道运行的 **max-keep-run** 值的最大值。	不适用
default-max-keep-runs	管道运行的 **max-keep-run** 值的默认限制。如果定义，该值将应用到没有 **max-keep-run** 注解的所有管道运行。	不适用
auto-configure-new-github-repo	自动配置新的 GitHub 存储库。Pipelines as Code 设置命名空间，并为存储库创建一个自定义资源。这个参数只支持 GitHub 应用程序。	**disabled**
auto-configure-repo-namespace-template	如果启用了 **auto-configure-new-github-repo**，将模板配置为为新仓库自动生成命名空间。	**{repo_name}-pipelines**
error-log-snippet	启用或禁用失败任务的日志片断视图，管道中有一个错误。当管道的数据泄漏时，您可以禁用此参数。	**enabled**

4.8.20. Pipelines as Code 命令参考

tkn pac CLI 工具提供以下功能：

- Bootstrap Pipelines as Code 安装和配置。figuration.
- 创建一个新的 Pipelines as Code 仓库。
- 列出所有 Pipelines as Code 仓库。
- 描述一个 Pipelines as Code 仓库和相关联的运行。
- 生成简单的管道运行以开始。

- 如由 Pipelines as Code 执行来解析一个管道运行。

提示

您可以使用与功能对应的命令进行测试和试验，因此您不必对包含应用源代码的 Git 仓库进行更改。

4.8.20.1. 基本语法

```
$ tkn pac [command or options] [arguments]
```

4.8.20.2. 全局选项

```
$ tkn pac --help
```

4.8.20.3. 工具命令

4.8.20.3.1. bootstrap

表 4.9. 将管道作为代码安装和配置引导

命令	描述
tkn pac bootstrap	安装并配置 Pipelines 作为 Git 仓库托管服务提供商的 Code，如 GitHub 和 GitHub Enterprise。
tkn pac bootstrap --nightly	安装每天（nightly）构建的 Pipelines as Code。
tkn pac bootstrap --route-url <public_url_to_ingress_spec>	覆盖 OpenShift 路由 URL。 默认情况下，**tkn pac bootstrap** 会检测 OpenShift 路由，该路由会自动与 Pipelines as Code 控制器服务关联。 如果您没有 OpenShift Container Platform 集群，它会要求您输入指向入口端点的公共 URL。
tkn pac bootstrap github-app	在 **pipelines-as-code** 命名空间中创建 GitHub 应用程序和 secret。

4.8.20.3.2. 软件仓库

表 4.10. 管理 Pipelines 作为代码软件仓库

命令	描述
tkn pac repo create	根据管道运行模板创建一个新的 Pipelines as Code 仓库以及一个命名空间。

命令	描述
tkn pac repo list	列出所有 Pipelines as Code 软件仓库，并显示关联运行的最后一个状态。
tkn pac repo describe	描述一个 Pipelines as Code 仓库和相关联的运行。

4.8.20.3.3. generate

表 4.11. 使用 Pipelines as Code 创建管道运行

命令	描述
tkn pac generate	生成简单的管道运行。 从包含源代码的目录执行时，它会自动检测当前的 Git 信息。 另外，它使用基本的语言检测功能，并根据语言添加额外的任务。 例如，如果它在仓库的 root 中检测到一个 **setup.py** 文件，则 pylint 任务会自动添加到生成的管道运行中。

4.8.20.3.4. 解析

表 4.12. 使用 Pipelines as Code 解析并执行管道运行

命令	描述
tkn pac resolve	执行管道运行，就像由 Pipelines as Code 服务中所有的一样。
tkn pac resolve -f .tekton/pull-request.yaml | oc apply -f -	显示在 **.tekton/pull-request.yaml** 中使用模板的实时管道运行状态。 结合在本地机器上运行的 Kubernetes 安装，您可以在不生成新提交的情况下观察管道运行。 如果从源代码存储库运行命令，它会尝试检测当前的 Git 信息并自动解析当前修订或分支等参数。

命令	描述
tkn pac resolve -f .tekton/pr.yaml -p revision=main -p repo_name= <repository_name>	通过覆盖从 Git 存储库派生的默认参数值来执行管道运行。 **-f** 选项也可以接受目录路径，并在该目录中的所有 **.yaml** 或 **.yml** 文件中应用 **tkn pac resolve** 命令。您还可以在同一命令中多次使用 **-f** 标志。 您可以使用 **-p** 选项指定参数值，覆盖 Git 仓库收集的默认信息。例如，您可以使用 Git 分支作为修订和不同的仓库名称。

4.8.21. 其他资源

- Pipelines as Code 存储库中的 **.tekton/** 目录示例
- 安装 OpenShift Pipelines
- 安装 tkn
- Red Hat OpenShift Pipelines 发行注记

4.9. 在 WEB 控制台中使用 RED HAT OPENSHIFT PIPELINES

您可以使用 **Administrator** 或 **Developer** 视角从 OpenShift Container Platform Web 控制台中的 **Pipelines** 页面创建和修改 **Pipeline**、**PipelineRun** 和 **Repository** 对象。您还可以使用 web 控制台的 **Developer** 视角中的 **+Add** 页面为软件交付过程创建 CI/CD 管道。

4.9.1. 在 Developer 视角中使用 Red Hat OpenShift Pipelines

在 **Developer** 视角中，您可以从 **+Add** 页面访问以下选项来创建管道：

- 使用 **+Add → Pipelines → Pipeline builder** 选项为您的应用程序创建自定义管道。
- 使用 **+Add → From Git** 选项，在创建应用程序时使用管道模板和资源创建管道。

在为应用程序创建管道后，可以在 **Pipelines** 视图中查看并以视觉化的形式与部署进行交互。您还可以使用 **Topology** 视图与使用 **From Git** 选项创建的管道交互。您需要将自定义标识应用到使用 **Pipeline builder** 创建的管道，以便在 **Topology** 视图中查看它。

先决条件

- 您可以访问 OpenShift Container Platform 集群，并切换到 **Developer** 视角。
- 在集群中安装了 Pipelines Operator。
- 您是集群管理员，或是有创建和编辑权限的用户。
- 您已创建了一个项目。

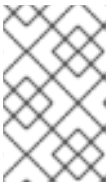
4.9.2. 使用 Pipeline 构建器构建管道

在控制台的 **Developer** 视角中，您可以使用 **+Add → Pipeline → Pipeline Builder** 选项：

- 使用 **Pipeline 构建器** 或 **YAML 视图** 配置管道。
- 使用现有任务和集群任务构建管道流。安装 OpenShift Pipelines Operator 时，它会在集群中添加可重复使用的管道集群任务。
- 指定管道运行所需的资源类型，如有必要，将额外参数添加到管道。
- 引用管道中的每个任务中的这些管道资源作为输入和输出资源。
- 如果需要，引用任务中添加至管道的任何额外参数。任务的参数会根据任务的规格预先填充。
- 使用 Operator 安装的、可重复使用的片断和示例来创建详细的管道。

流程

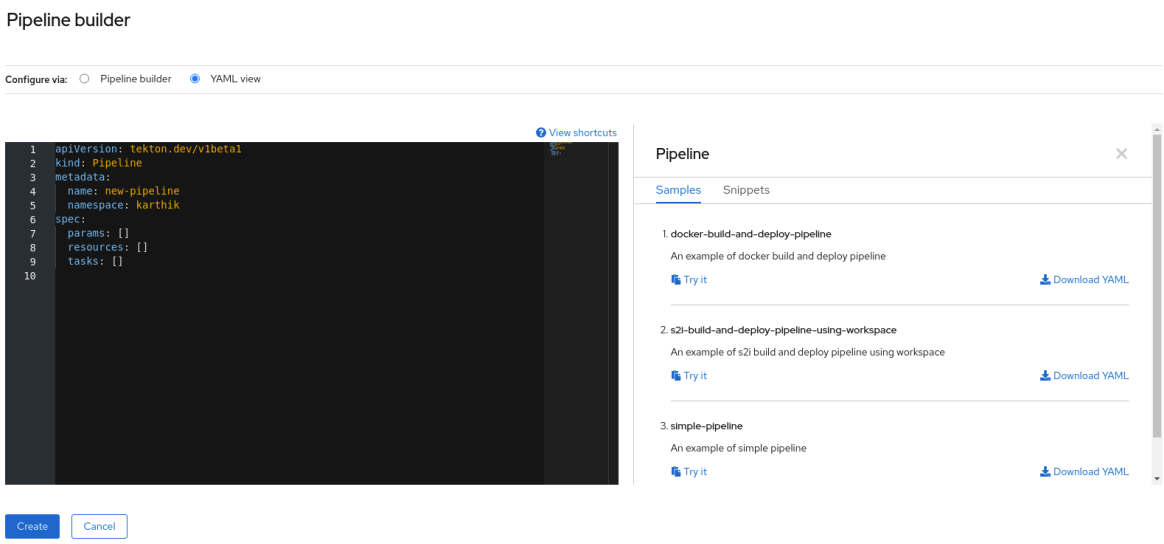
1. 在 **Developer** 视角的 **Add** 视图中，点 **Pipeline** 标题查看 **Pipeline Builder** 页面。
2. 使用 **Pipeline 构建器** 视图或 **YAML 视图**配置管道。



注意

Pipeline 构建器 视图支持有限的字段，而 **YAML 视图**支持所有可用字段。（可选）您还可以使用 Operator 安装的、可重复使用的代码片断和样本来创建详细的管道。

图 4.1. YAML 视图



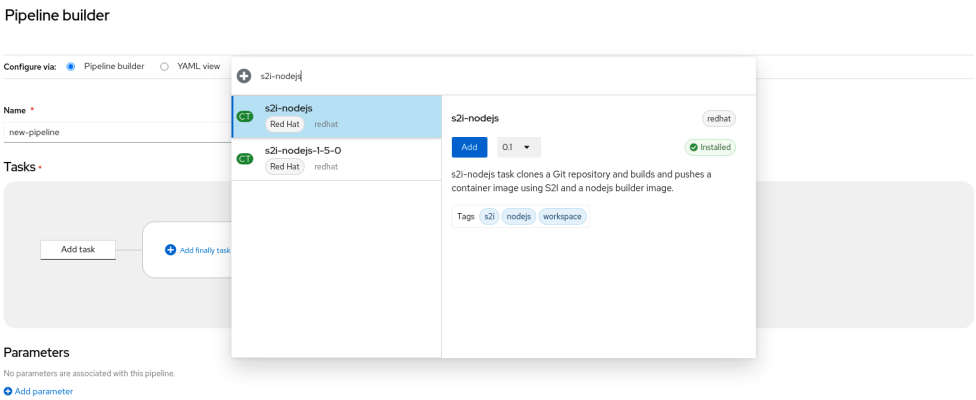
3. 使用 **Pipeline 构建器**配置管道：
   a. 在 **Name** 字段中输入管道的唯一名称。
   b. 在 **Tasks** 部分：
      i. 单击 **Add task**。
      ii. 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
      iii. 点击 **Add** 或 **Install and add**。在本例中,使用 **s2i-nodejs** 任务。

注意

搜索列表包含集群中可用的所有 Tekton Hub 任务和任务。此外，如果某一任务已经安装，它将显示 **Add** 添加该任务，而它将显示 **Install and add** 以安装和添加该任务。当使用更新的版本添加相同的任务时，它将显示 **Update and add**。

- 在管道中添加后续任务：
  - 点击任务右侧或左侧的加号图标 → 点 **Add task**。
  - 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
  - 点击 **Add** 或 **Install and add**。

  图 4.2. Pipeline 构建器



- 添加最后一项任务：
  - 点 **Add finally task** → Click **Add task**。
  - 使用快速搜索字段搜索任务，然后从显示的列表中选择所需的任务。
  - 点击 **Add** 或 **Install and add**。

c. 在 **Resources** 部分，点 **Add Resources** 指定管道运行的资源的名称和类型。这些资源然后会被管道中的任务使用作为输入和输出。在此例中：

   i. 添加一个输入资源。在 **Name** 字段中输入 **Source**，从 **Resource Type** 下拉列表中选择 **Git**。

   ii. 添加一个输出资源。在 **Name** 字段中输入 **img**，从 **Resource Type** 下拉列表中选择 **Image**。

注意

如果缺少资源，任务旁边会出现一个红色图标。

d. 可选：任务的 **Parameters** 部分会根据任务的规格预先填充。如果需要，使用 **Parameters** 部分中的 **Add Parameters** 链接来添加额外的参数。

e. 在 **Workspaces** 部分，点 **Add workspace**，并在 **Name** 字段中输入唯一工作区名称。您可以在管道中添加多个工作区。

f. 在 **Tasks** 部分中，点 **s2i-nodejs** 任务，以查看带任务详情的侧面板。在任务侧面板中，为 **s2i-nodejs** 任务指定资源和参数：

i. 如果需要，在 **Parameters** 部分，使用 $(params.<param-name>)语法为默认参数添加更多参数。

ii. 在 **Image** 部分中，按在 **Resources** 部分中的指定输入 **Img**。

iii. 从 **Workspaces** 部分下的 **source** 下拉菜单中选择一个工作区。

g. 将资源、参数和工作空间添加到 **openshift-client** 任务。

4. 点 **Create** 在 **Pipeline Details** 页面中创建并查看管道。

5. 点 **Actions** 下拉菜单，然后点 **Start**，查看 **Start Pipeline** 页面。

6. **Workspaces** 部分列出了您之前创建的工作区。使用对应的下拉菜单为您的工作区指定卷源。您有以下选项： **mpty Directory**、**Config Map**、**Secret**、**PersistentVolumeClaim** 或 **VolumeClaimTemplate**。

4.9.3. 创建 OpenShift Pipelines 和应用程序

要与应用程序一同创建管道，使用 **Developer** 视角的 **Add+** 视图中的 **From Git** 选项。您可以查看所有可用的管道，并在导入代码或部署镜像时选择用来创建应用程序的管道。

Tekton Hub 集成默认是启用的，您可以看到集群支持的 Tekton Hub 中的任务。管理员可以选择不使用 Tekton Hub 集成，将不会显示 Tekton Hub 任务。您还可以检查是否为生成的管道存在 webhook URL。为使用 **+Add** 流创建的管道添加默认 Webhook，URL 在 Topology 视图中所选资源侧面面板中可见。

如需更多信息，请参阅使用 Developer 视角创建应用程序。

4.9.4. 使用 Developer 视角与管道交互

开发者视角中的 **Pipelines** 视图列出了项目中的所有管道，以及以下详细信息：

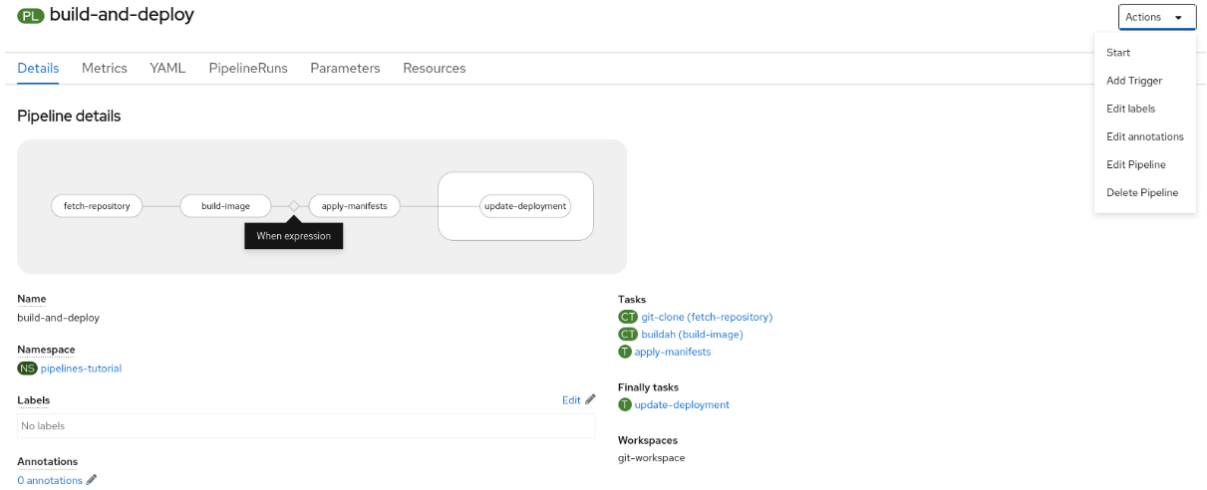
- 创建管道的命名空间
- 最后一次管道运行
- 管道运行中的任务状态
- 管道运行的状态
- 最后一次管道运行的创建时间

流程

1. 在 **Developer** 视角的 **Pipelines** 视图中，从 **Project** 下拉列表中选择一个项目，以查看该项目中的管道。

2. 点击所需管道查看 **Pipeline 详情**页面。
默认情况下，**Details** 选项卡显示所有串行任务、并行任务、**finally** 任务以及管道中的 when 表达式的可视化表示。这些任务和 **finally** 的任务列在页面的右下角。点列出的 **Tasks** 和 **Finally tasks** 以查看任务详情。

图 4.3. Pipeline 详情



3. 可选： 在 **Pipeline 详情**页面中，点 **Metrics** 选项卡查看有关管道的以下信息：
   - **Pipeline 成功率**
   - **Pipeline 运行数量**
   - **管道运行持续时间**
   - **任务运行持续时间**
     您可以使用这些信息来改进管道工作流，并在管道生命周期的早期解决问题。
4. 可选：点 **YAML** 选项卡编辑管道的 YAML 文件。
5. 可选：点 **Pipeline Runs** 选项卡查看已完成、正在运行或运行失败的管道。
   Pipeline Runs 选项卡提供有关管道运行、任务状态以及调试失败管道运行的链接。您可以使用 Options 菜单 来停止正在运行的管道，使用与之前的管道执行相同的参数和资源重新运行管道，或删除管道运行。
   - 点所需的管道运行查看 **Pipeline Run details** 页面。默认情况下，**Details** 选项卡显示所有串行任务、并行任务、**finally** 任务以及管道中运行的 when 表达式的可视化表示。成功运行的结果会显示在页面底部的 **Pipeline Run 结果**窗格下。另外，您只能查看由集群支持的 Tekton Hub 中的任务。在查看某个任务时，您可以点击相关链接来跳至任务文档。

     注意

     Pipeline Run Details 页面的 **Details** 部分显示失败管道运行的**日志片段**。**日志片断** 提供一般错误信息和日志片断。**Logs** 部分的链接可让您快速访问失败运行的详细信息。
   - 在 **Pipeline Run details** 页面中，点 **Task Runs** 选项卡查看已完成、正在运行和运行失败的任务。
     Task Runs 选项卡提供有关任务运行的信息，以及任务和 pod 的链接，以及任务运行的状态和持续时间。使用 Options 菜单 来删除任务运行。
   - 点所需的任务运行查看 **Task Run details** 页面。成功运行的结果显示在页面底部的 **Task Run 结果**窗格下。

注意

Task Run details页面的 **Details** 部分显示失败任务运行的**日志片段**。**日志片断** 提供一般错误信息和日志片断。**Logs** 部分的链接可让您快速访问失败的任务运行的详细信息。

6. 点 **Parameters** 标签，查看管道中定义的参数。您还可以根据需要添加或者编辑附加参数。
7. 点 **Resources** 标签页，查看管道中定义的资源。您还可以根据需要添加或编辑附加资源。

4.9.5. 从 Pipelines 视图启动管道

创建管道后，您需要启动它以在定义的顺序中执行包含的任务。您可从 **Pipelines** 视图、**Pipeline Details** 页面或 **Topology** 视图启动管道。

流程

使用 **Pipelines** 视图启动管道：

1. 在 **Developer 视角** 的 **Pipelines** 视图中，点附加到 Pipeline 的 **Options**  菜单，然后选择 **Start**。
2. **Start Pipeline**对话框显示 **Git Resources** 以及基于管道定义的 **Image Resources**。

注意

对于使用 **From Git**选项创建的管道，**Start Pipeline**对话框也会在 **Parameters** 部分显示 **APP_NAME** 字段，对话框中的所有字段都由管道模板预先填充。

   a. 如果您在命名空间中有资源，**Git Resources** 和 **Image Resources** 字段会预先填充这些资源。如果需要，使用下拉菜单选择或创建所需资源并自定义管道运行实例。

3. 可选：修改 **Advanced Options** 以添加验证指定私有 Git 服务器或镜像 registry 的凭证。
   a. 在 **Advanced Options** 下，点 **Show Credentials Options** 并选择 **Add Secret**。
   b. 在 **Create Source Secret**部分，指定以下内容：
      i. secret 的唯一 **Secret Name**。
      ii. 在 **要被验证的指定供应商** 部分，在 **Access to** 字段中指定要验证的供应商，以及基本 **服务器 URL**。
      iii. 选择 **Authentication Type** 并提供凭证：
         - 对于 **Authentication Type Image Registry Credentials**，请指定您要进行身份验证的 **Registry 服务器地址**，并通过**Username**、**Password** 和 **Email** 项中提供您的凭证。
           如果要指定额外的 **Registry 服务器地址**，选择 **Add Credentials**。
         - 如果 **Authentication Type** 为 **Basic Authentication**，在 **UserName** 和 **Password or Token** 项中指定相关的值。
         - 如果 **Authentication Type** 为 **SSH Keys** 时，在 **SSH Private Key** 字段中指定相关的值。

注意

对于基本身份验证和 SSH 身份验证，您可以使用注解，例如：

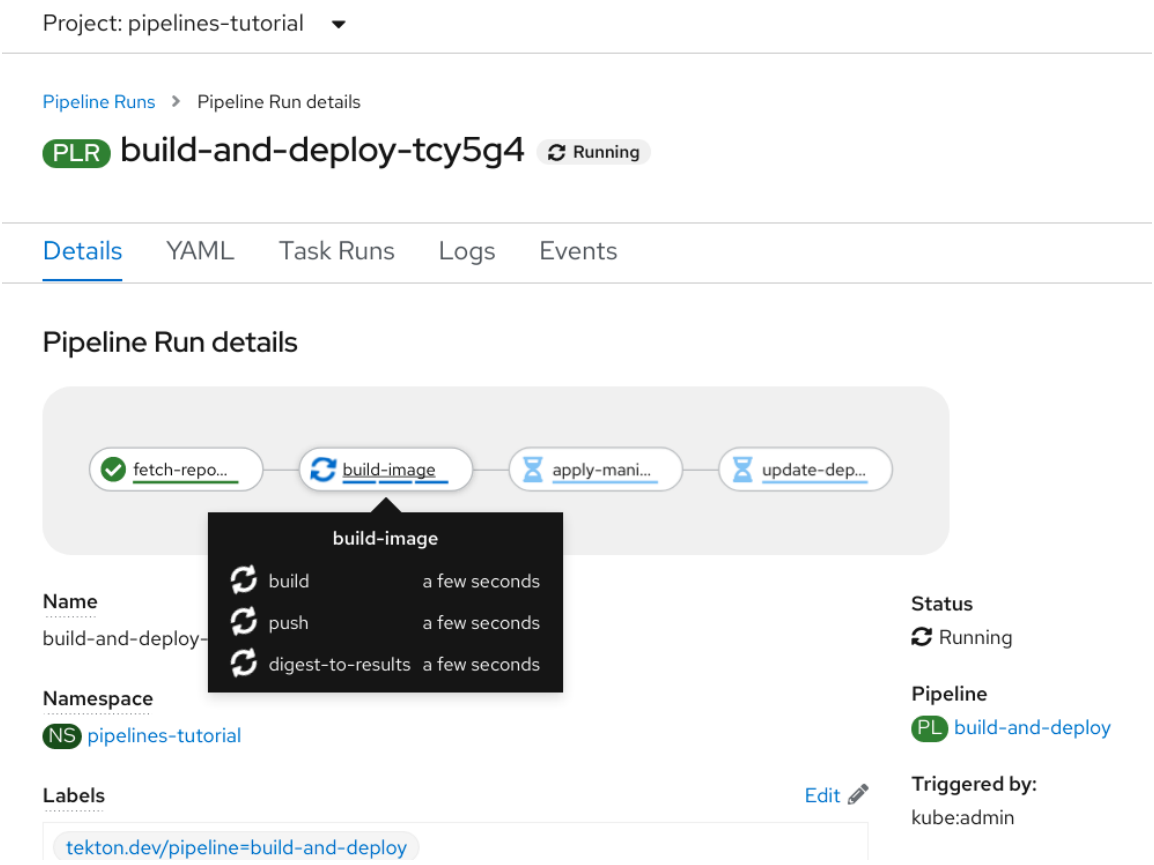
- **tekton.dev/git-0: https://github.com**
- **tekton.dev/git-1: https://gitlab.com**.

iv. 选择要添加 secret 的检查标记。

您可以根据频道中的资源数量添加多个 secret。

4. 点 **Start** 启动管道。
5. **Pipeline Run Details** 页面显示正在执行的管道。管道启动后，每个任务中的任务和步骤都会被执行。您可以：
   - 将鼠标悬停在任务上，以查看执行每一步骤所需时间。
   - 点一个任务来查看任务中每一步骤的日志。
   - 点 **Logs** 选项卡查看与任务执行顺序相关的日志。您还可以使用相关按钮扩展窗格，单独或批量下载日志。
   - 点 **Events** 选项卡查看管道运行生成的事件流。
     您可以使用 **Task Runs**、**Logs** 和 **Events** 选项卡来帮助调试失败的管道运行或失败的任务运行。

图 4.4. Pipeline 运行详情



4.9.6. 从 Topology 视图启动管道

对于使用 **From Git** 选项创建的管道，您可以在启动后使用 **Topology** 视图来与管道进行交互：

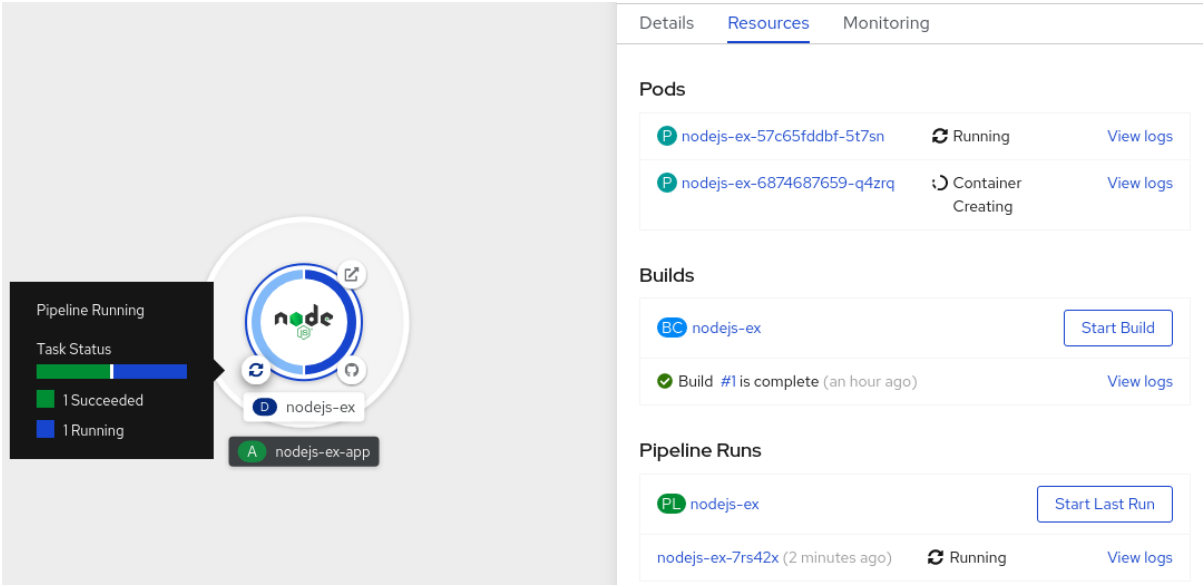
注意

要使用 **Pipeline Builder** 在 **Topology** 视图中查看创建的管道，自定义管道标识来把 Pipeline 与应用程序负载相连接。

流程

1. 单击左侧导航面板中的 **Topology**。
2. 点应用程序在侧面面板中显示 **Pipeline Runs**。
3. 在 **Pipeline Runs** 中，点 **Start Last Run** 来启动一个新的管道运行，使用与前一个相同的参数和资源。如果没有启动管道运行，这个选项会被禁用。您还可以在创建管道时启动管道运行。

图 4.5. Topology 视图中的管道



在 **Topology** 页面中，把鼠标移到应用程序的左侧，以查看其管道运行的状态。添加管道后，底部左图标表示存在关联的管道。

4.9.7. 从 Topology 视图与管道交互

Topology 页面中的应用程序节点的侧面板显示管道运行的状态，并可与之交互。

- 如果管道运行没有自动启动，则侧面板会显示管道无法自动启动的消息，因此需要手动启动。
- 如果创建一个管道，但用户没有启动管道，则其状态不会启动。当用户点击 **Not started** status 图标时，**Topology** 视图中会打开启动对话框。
- 如果管道没有构建或构建配置，则无法看到 **Builds** 部分。如果存在管道和构建配置，则 **Builds 部分** 可见。
- 当管道运行在特定任务运行时，侧边面板会显示一个 **日志片段**。您可以在 **Pipeline Runs** 部分的 **Resources** 选项卡中查看**日志片断**。它提供常规错误消息和日志片断。**Logs** 部分的链接可让您快速访问失败运行的详细信息。

4.9.8. 编辑管道

您可以使用 web 控制台的 **Developer** 视角编辑集群中的 Pipelines:

流程

1. 在 **Developer** 视角的 **Pipelines** 视图中，选择您要编辑的管道来查看 Pipeline 的详情。在 **Pipeline Details** 页中，点 **Actions** 并选择 **Edit Pipeline**。
2. 在 **Pipeline builder** 页面中，您可以执行以下任务：
   - 在 Pipeline 中添加额外的任务、参数或资源。
   - 点要修改的任务来查看侧面面板中的任务详情，并修改所需的任务详情，如显示名称、参数和资源。
   - 或者，要删除此任务，点任务，在侧面面板中点 **Actions**，并选择 **Remove Task**。
3. 点 **Save** 来保存修改的管道。

4.9.9. 删除管道

您可以使用 web 控制台的 **Developer** 视角删除集群中的管道。

流程

1. 在 **Developer 视角** 的 **Pipelines** 视图中，点 Pipeline 旁的 **Options**  菜单，然后选择 **Delete Pipeline**。
2. 在 **Delete Pipeline** 确认提示下，点 **Delete** 以确认删除。

4.9.9.1. 其他资源

- 在 Pipelines 中使用 Tekton Hub

4.9.10. 在 Administrator 视角中创建管道模板

作为集群管理员，您可以创建管道模板，开发人员可在集群上创建管道时重复使用该模板。

先决条件

- 您可以使用集群管理员权限访问 OpenShift Container Platform 集群，并切换到 **Administrator** 视角。
- 已在集群中安装了 Pipelines Operator。

流程

1. 进入到 **Pipelines** 页面来查看现有的管道模板。
2. 点  图标进入 **Import YAML** 页面。
3. 为管道模板添加 YAML。模板必须包含以下信息：

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
# ...
  namespace: openshift (1)
  labels:
    pipeline.openshift.io/runtime: <runtime> (2)
    pipeline.openshift.io/type: <pipeline-type> (3)
# ...
```

1. 模板必须在 **openshift** 命名空间中创建。
2. 模板必须包含 **pipeline.openshift.io/runtime** 标签。此标签接受的运行时值为 **nodejs**,**golang**,**dotnet**,**java**,**php**, ruby ,**ruby**,**perl**,**python**,**nginx**, 和 **httpd**。
3. 模板必须包含 **pipeline.openshift.io/type:** 标签。此标签接受的 type 值为 **openshift**,**knative**, 和 **kubernetes**。

4. 点 **Create**。创建管道后，会进入 **Pipeline 详情页面**，您可以在其中查看或编辑管道的信息。

4.10. 在 TEKTONCONFIG 自定义资源中自定义配置

在 Red Hat OpenShift Pipelines 中，您可以使用 **TektonConfig** 自定义资源 (CR) 自定义以下配置：

- 配置 Red Hat OpenShift Pipelines control plane
- 更改默认服务帐户
- 禁用服务监控器
- 禁用集群任务和管道模板
- 禁用 Tekton Hub 的集成
- 禁用自动创建 RBAC 资源
- 修剪任务运行和管道运行

4.10.1. 先决条件

- 已安装 Red Hat OpenShift Pipelines Operator。

4.10.2. 配置 Red Hat OpenShift Pipelines control plane

您可以通过编辑 **TektonConfig** 自定义资源 (CR) 中的配置字段来自定义 Pipelines control plane。Red Hat OpenShift Pipelines Operator 会自动使用默认值添加配置字段，以便您可以使用 Pipelines control plane。

流程

1. 在 Web 控制台的 **Administrator** 视角中，进入到 **Administration → CustomResourceDefinitions**。

2. 使用 **Search by name** 复选框搜索 **tektonconfigs.operator.tekton.dev** 自定义资源定义 (CRD)。点 **TektonConfig** 查看 CRD 详情页面。

3. 点 **实例** 选项卡。

4. 点 **config** 实例查看 **TektonConfig** CR 详情。

5. 点 **YAML** 标签。

6. 根据您的要求编辑 **TektonConfig** YAML 文件。

带有默认值的 TektonConfig CR 示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    running-in-environment-with-injected-sidecars: true
    metrics.taskrun.duration-type: histogram
    metrics.pipelinerun.duration-type: histogram
    await-sidecar-readiness: true
    params:
      - name: enableMetrics
        value: 'true'
    default-service-account: pipeline
    require-git-ssh-secret-known-hosts: false
    enable-tekton-oci-bundles: false
    metrics.taskrun.level: task
    metrics.pipelinerun.level: pipeline
    embedded-status: both
    enable-api-fields: stable
    enable-provenance-in-status: false
    enable-custom-tasks: true
    disable-creds-init: false
    disable-affinity-assistant: true
```

4.10.2.1. 带有默认值的可修改字段

以下列表包含 **TektonConfig** CR 中所有可修改字段的默认值：

- **running-in-environment-with-injected-sidecars** （默认为 **true**）：如果管道在不使用注入的 sidecar 的集群中运行的管道（如 Istio）将此字段设置为 **false**。将它设置为 **false** 可缩短管道运行启动所需的时间。

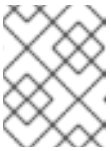
注意

对于使用注入的 sidecar 的集群，将此字段设置为 **false** 可能会导致意外行为。

- **await-sidecar-readiness** (默认： **true**): 将此字段设置为 **false** 以阻止 Pipelines 在开始操作前等待 **TaskRun** sidecar 容器运行。这允许在不支持 **downwardAPI** 卷类型的环境中运行任务。

- **default-service-account** （默认： **pipeline**）：此字段包含用于 **TaskRun** 和 **PipelineRun** 资源的默认服务帐户名称，如果没有指定。

- **require-git-ssh-secret-known-hosts** （默认为 **false**）：将此字段设置为 **true** 需要任何 Git SSH secret 都必须包含 **known_hosts** 字段。
  - 有关配置 Git SSH secret 的更多信息，请参阅*附加资源*部分中的*为 Git 配置 SSH 身份验证*。
- **enable-tekton-oci-bundles** (默认： **false**): 将此字段设置为 **true** 以启用名为 Tekton OCI 捆绑包的实验性 alpha 功能。
- **embedded-status**（默认为 **both**）：此字段有三个可接受值：
  - **full**: 启用在 **PipelineRun** 状态中嵌入的所有 **Run** 和 **TaskRun** 状态
  - **minimal** ：使用名称、类型和 API 版本等信息为 'PipelineRun' 状态中的每个运行和任务运行生成 **ChildReferences** 字段的信息。
  - **both**: 引用 **full** 和 **minimal** 两者的值



注意

embedded-status 字段已弃用，并将在以后的发行版本中删除。另外，管道默认嵌入状态将改为 **minimal**。

- **enable-api-fields** （默认：**stable**）：设置此字段决定启用哪些功能。可接受的值为 **stable**、**beta** 或 **alpha**。

注意

Red Hat OpenShift Pipelines 不支持 **alpha** 值。

- **enable-provenance-in-status** （默认：**false**）：将此字段设置为 **true** 以启用 **TaskRun** 和 **PipelineRun** 状态中的 **provenance** 字段。**provenance** 字段包含有关任务运行和管道运行中使用的资源的元数据，如从中获取远程任务或管道定义的源。
- **enable-custom-tasks** （默认为 **true**）：将此字段设置为 **false**，以禁用管道中的自定义任务。
- **disable-creds-init** （默认为 **false**）：将此字段设置为 **true**，以防止 Pipelines 扫描附加的服务帐户并将任何凭证注入步骤。
- **disable-affinity-assistant** （默认为 **true**）：将此字段设置为 **false**，为每个共享持久性卷声明工作区的 **TaskRun** 资源启用关联性协助功能。

指标选项

您可以在 **TektonConfig** CR 中修改以下 metrics 字段的默认值：

- **metrics.taskrun.duration-type** 和 **metrics.pipelinerun.duration-type** (默认：**histogram**)：设置这些字段决定了任务或管道运行的持续时间类型。可接受的值是 **gauge** 或 **histogram**。
- **metrics.taskrun.level** (默认：**task**)：此字段决定了任务运行指标的级别。可接受的值为 **taskrun**, **task**, 或 **namespace**。
- **metrics.pipelinerun.level** （默认为 **pipeline**）：此字段决定了管道运行指标的级别。可接受的值为 **pipelinerun**、**pipeline** 或 **namespace**。

4.10.2.2. 可选的配置字段

以下字段没有默认值，只有在配置它们时才会考虑它们。默认情况下，Operator 不会在 **TektonConfig** 自定义资源 (CR) 中添加和配置这些字段。

- **default-timeout-minutes** ：此字段为 **TaskRun** 和 **PipelineRun** 资源设置默认超时时间（如果没有在创建它们时指定）。如果任务运行或管道运行需要的时间超过其执行设置的分钟数，则任务运行或管道运行会超时并取消。例如，**default-timeout-minutes: 60** 代表将默认值设为 60 分钟。
- **default-managed-by-label-value** ：此字段包含提供给 **app.kubernetes.io/managed-by** 标签的默认值，如果未指定，它被应用到所有 **TaskRun** pod。例如，**default-managed-by-label-value: tekton-pipelines**。
- **default-pod-template** ：此字段设置默认 **TaskRun** 和 **PipelineRun** pod 模板（如果没有指定）。
- **default-cloud-events-sink** ：此字段设置**TaskRun** 和 **PipelineRun** 资源的默认 **CloudEvents** sink（如果未指定）。
- **default-task-run-workspace-binding** ：此字段包含 **Task** 资源已声明，但 **TaskRun** 资源没有明确声明的工作区的默认工作区配置。
- **default-affinity-assistant-pod-template** ：此字段设置用于关联性助手 pod 的默认 **PipelineRun** pod 模板（如果没有指定）。
- **default-max-matrix-combinations-count** ：此字段包含从列表中生成的默认最大组合数（如果未指定）。

4.10.3. 更改 Pipelines 的默认服务帐户

您可以通过编辑 **.spec.pipeline** 和 **.spec.trigger** 规格中的 **default-service-account** 字段来更改 Pipelines 的默认服务帐户。默认服务帐户名称为 **pipeline**。

示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    default-service-account: pipeline
  trigger:
    default-service-account: pipeline
    enable-api-fields: stable
```

4.10.4. 禁用服务监控器

您可以禁用作为 Pipelines 一部分的服务监控器，以公开遥测数据。要禁用服务监控器，请在 **TektonConfig** 自定义资源 (CR) 的 **.spec.pipeline** 规格中将 **enableMetrics** 参数设置为 **false** ：

示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
```

```
  name: config
spec:
  pipeline:
    params:
      - name: enableMetrics
        value: 'false'
```

4.10.5. 禁用集群任务和管道模板

默认情况下，**TektonAddon** 自定义资源 (CR) 在集群中安装 **clusterTasks** 和 **pipelineTemplates** 资源，以及集群中的 Pipelines。

您可以通过在 **.spec.addon** 规格中将参数值设置为 **false** 来禁用 **clusterTasks** 和 **pipelineTemplates** 资源的安装。另外，您可以禁用 **communityClusterTasks** 参数。

示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  addon:
    params:
      - name: clusterTasks
        value: 'false'
      - name: pipelineTemplates
        value: 'false'
      - name: communityClusterTasks
        value: 'true'
```

4.10.6. 禁用 Tekton Hub 的集成

您可以通过在 **TektonConfig** 自定义资源 (CR) 中将 **enable-devconsole-integration** 参数设置为 **false** 来禁用 web 控制台 **Developer** 视角中的 Tekton Hub 集成。

禁用 Tekton Hub 示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  hub:
    params:
      - name: enable-devconsole-integration
        value: false
```

4.10.7. 禁用自动创建 RBAC 资源

Red Hat OpenShift Pipelines Operator 的默认安装会为集群中的所有命名空间创建多个基于角色的访问控制(RBAC)资源，但与 **^(openshift|kube)-*** 正则表达式模式匹配的命名空间除外。在这些 RBAC 资源中，**pipelines-scc-rolebinding** 安全性上下文约束(SCC)角色绑定资源是一个潜在的安全问题，因为关联的 **pipelines-scc** SCC 具有 **RunAsAny** 特权。

要在安装 Red Hat OpenShift Pipelines Operator 后禁用集群范围的 RBAC 资源，集群管理员可在集群级 **TektonConfig** 自定义资源(CR)中将 **createRbacResource** 参数设置为 **false**。

TektonConfig CR 示例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
  - name: createRbacResource
    value: "false"
...
```

警告

作为集群管理员或具有适当权限的用户，当您为所有命名空间禁用自动创建 RBAC 资源时，默认的 **ClusterTask** 资源无法正常工作。要使 **ClusterTask** 资源正常工作，您必须手动为每个预期的命名空间创建 RBAC 资源。

4.10.8. 自动修剪任务运行和管道运行

过时的 **TaskRun** 和 **PipelineRun** 对象及其执行的实例占用了可用于活跃运行的物理资源。为了优化这些资源，Red Hat OpenShift Pipelines 提供了注解，集群管理员可以使用它们来自动修剪不同命名空间中未使用的对象及其实例。

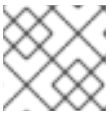
注意

通过指定注解配置自动修剪会影响整个命名空间。您不能选择性地自动修剪一个命名空间中的单个任务运行或管道运行。

4.10.8.1. 自动修剪任务运行和管道运行的注解

要自动修剪任务运行和管道运行，您可以在命名空间中创建以下注解：

- **operator.tekton.dev/prune.schedule**：如果此注解的值与 **TektonConfig** 自定义资源定义中指定的值不同，则会在该命名空间中创建新的 cron 作业。
- **operator.tekton.dev/prune.skip**：设置为 **true** 时，配置它的命名空间不会被修剪。
- **operator.tekton.dev/prune.resources**：此注解接受以逗号分隔的资源列表。要修剪单一资源，如管道运行，将此注解设置为 **"pipelinerun"**。要修剪多个资源，如任务运行和管道运行，将此注解设置为 **"taskrun, pipelinerun"**。
- **operator.tekton.dev/prune.keep**：使用此注解保留资源而不进行修剪。
- **operator.tekton.dev/prune.keep-since**：使用此注解来根据其存在的时间来保留资源。此注解的值必须等于资源的年龄（以分钟为单位）。例如，若要保留在五天内创建的资源，请将 **keep-since** 设置为 **7200**。



注意

keep 和 **keep-since** 注释是互斥的。对于任何资源，您只能配置其中一个。

- **operator.tekton.dev/prune.strategy**：将此注解的值设置为 **keep** 或 **keep-since**。

例如，请考虑以下注解，这些注解会保留在最后五天中创建的所有任务运行和管道运行，并删除旧的资源：

auto-pruning 注解示例

```
...
  annotations:
    operator.tekton.dev/prune.resources: "taskrun, pipelinerun"
    operator.tekton.dev/prune.keep-since: 7200
...
```

4.10.9. 其他资源

- 为 Git 配置 SSH 身份验证
- 管理未指定版本的和版本化的集群任务
- 修剪对象以重新声明资源
- 在 Administrator 视角中创建管道模板

4.11. 减少 OPENSHIFT PIPELINES 的资源消耗

如果在多租户环境中使用集群，您必须控制各个项目和 Kubernetes 对象的 CPU、内存和存储资源的消耗。这有助于防止一个应用程序消耗太多资源并影响其他应用程序。

要定义在生成的 pod 上设置的最终资源限值，Red Hat OpenShift Pipelines 使用资源配额限制和/或限制执行项目的范围。

要限制项目中的资源消耗，您可以：

- 设置和管理资源配额，以限制聚合资源消耗。
- 使用限制范围来限制特定对象的资源消耗，如 pod、镜像、镜像流和持久性卷声明。

4.11.1. 了解管道中的资源消耗

每个任务都由 **Task** 资源中 **steps** 字段中定义的特定顺序执行的一些所需步骤组成。每个任务都作为 pod 运行，每个步骤都作为该 pod 中的容器运行。

每次都会执行一个步骤。执行此任务的 Pod 仅请求足够的资源，以一次在任务中运行单个容器镜像（步骤），因此不会为任务中的所有步骤存储资源。

steps spec 中的 **Resources** 字段指定资源消耗的限值。默认情况下，CPU、内存和临时存储的资源请求设置为 **BestEffort**（零）值，或通过该项目中限制范围设置的最小值。

步骤的资源请求和限值配置示例

```
spec:
  steps:
  - name: <step_name>
    resources:
      requests:
        memory: 2Gi
        cpu: 600m
      limits:
        memory: 4Gi
        cpu: 900m
```

当在执行管道和任务运行的项目中指定 **LimitRange** 参数和容器资源请求的最小值时，Red Hat OpenShift Pipelines 会查看项目中的所有 **LimitRange** 值，并使用最小值而不是零。

在项目级别配置限制范围参数示例

```
apiVersion: v1
kind: LimitRange
metadata:
  name: <limit_container_resource>
spec:
  limits:
  - max:
      cpu: "600m"
      memory: "2Gi"
    min:
      cpu: "200m"
      memory: "100Mi"
    default:
      cpu: "500m"
      memory: "800Mi"
    defaultRequest:
      cpu: "100m"
      memory: "100Mi"
    type: Container
...
```

4.11.2. 缓解管道中的额外资源消耗

当您在 pod 中的容器上设置了资源限制时，OpenShift Container Platform 会将请求的资源限值作为同时运行的所有容器的总和。

为了消耗调用任务一次执行一个步骤所需的最小资源，Red Hat OpenShift Pipelines 请求最大 CPU、内存和临时存储，这在需要最多资源的步骤中指定。这样可确保满足所有步骤的资源要求。最大值以外的请求设置为零。

但是，这种行为的资源使用量会超过要求。如果使用资源配额，这也可能导致无法调度的 pod。

例如，有一个任务，它带有两个使用脚本的步骤，且没有定义任何资源限值和请求。生成的 pod 有两个 init 容器（一个用于入口点复制，另一个用于编写脚本）和两个容器（每个步骤一个）。

OpenShift Container Platform 使用为项目设置的限制范围来计算所需资源请求和限值。在本例中，在项目中设置以下限制范围：

```
apiVersion: v1
```

```
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
```

在这种情况下，每个 init 容器使用 1Gi 请求内存（限制范围的最大限制），每个容器都使用 500Mi 请求内存。因此，pod 的内存请求总数为 2Gi。

如果对一个有十个步骤的任务使用相同的限制范围，则最终内存请求为 5Gi，高于每个步骤实际需要的 500Mi（因为每个步骤均在另一个步骤之后运行）。

因此，为了减少资源消耗，您可以：

- 通过将不同的步骤分组到一个较大的步骤，使用脚本功能和同一镜像来减少给定任务中的步骤数量。这可减少最低请求资源。
- 分发步骤，它们相互独立，可以对于多个任务而不是单个任务运行。这样可减少每个任务中的步骤数量，使每个任务的请求更小，那么调度程序可在资源可用时运行它们。

4.11.3. 其他资源

- 为 OpenShift Pipelines 设置计算资源配额
- 项目的资源配额
- 使用限制范围限制资源消耗
- Kubernetes 中的资源请求和限值

4.12. 为 OPENSHIFT PIPELINES 设置计算资源配额

Red Hat OpenShift Pipelines 中的 **ResourceQuota** 对象控制每个命名空间的总资源消耗。您可以使用它来限制命名空间中创建的对象数量，具体取决于对象的类型。另外，您可以指定计算资源配额来限制命名空间中消耗的计算资源总量。

但是，您可能希望限制管道运行生成的 Pod 所消耗的计算资源数量，而不是为整个命名空间设置配额。目前，Red Hat OpenShift Pipelines 不会让您直接为管道指定计算资源配额。

4.12.1. 在 OpenShift Pipelines 中限制计算资源消耗的替代方法

要获得对管道使用计算资源的一定程度的控制，请考虑以下备选方法：

- 为任务中的每个步骤设置资源请求和限值。

  示例：为任务中的每个步骤设置资源请求和限值。

  ```
  ...
  spec:
    steps:
  ```

```
    - name: step-with-limts
      resources:
        requests:
          memory: 1Gi
          cpu: 500m
        limits:
          memory: 2Gi
          cpu: 800m
...
```

- 通过为 **LimitRange** 对象指定值来设置资源限值。如需有关 **LimitRange** 的更多信息，请参阅限制范围限制资源消耗。
- 减少管道资源消耗。
- 设置和管理 每个项目的资源配额。
- 理想情况下，管道的计算资源配额应与管道运行中同时运行的 Pod 所消耗的计算资源总量相同。不过，运行任务的容器集会根据用例消耗计算资源。例如，Maven 构建任务可能需要为它构建的不同应用提供不同的计算资源。因此，您无法为通用管道中的任务预先确定计算资源配额。要提高计算资源使用率的可预测性，并对计算资源的使用进行控制，请将自定义管道用于不同的应用程序。

注意

在配置了 **ResourceQuota** 对象的命名空间中使用 Red Hat OpenShift Pipelines 时，任务运行和管道运行的 pod 可能会失败，并显示错误，如：**failed quota: <quota name> must specify cpu, memory**。

要避免这个错误，请执行以下操作之一：

- （推荐）为命名空间指定一个限制范围。
- 为所有容器明确定义请求和限值。

如需更多信息，请参阅问题和解决方案。

如果您的用例没有通过这些方法解决，您可以使用优先级类的资源配额来实施临时解决方案。

4.12.2. 使用优先级类指定管道资源配额

PriorityClass 对象将优先级类名称映射到表示其相对优先级的整数值。数值越高，类的优先级越高。创建优先级类后，您可以创建 pod，在其规格中指定优先级类名称。另外，您可以根据 pod 优先级控制 pod 对系统资源的消耗。

为管道指定资源配额类似于为管道运行创建的 pod 子集设置资源配额。以下步骤通过根据优先级类指定资源配额来提供临时解决方案示例。

流程

1. 为管道创建优先级类。

   示例：管道的优先级类

   ```
   apiVersion: scheduling.k8s.io/v1
   ```

```
kind: PriorityClass
metadata:
  name: pipeline1-pc
value: 1000000
description: "Priority class for pipeline1"
```

2. 为管道创建资源配额。

   示例：管道的资源配额

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pipeline1-rq
spec:
  hard:
    cpu: "1000"
    memory: 200Gi
    pods: "10"
  scopeSelector:
    matchExpressions:
    - operator : In
      scopeName: PriorityClass
      values: ["pipeline1-pc"]
```

3. 验证管道的资源配额使用量。

   示例：验证管道的资源配额使用情况

```
$ oc describe quota
```

   输出示例

```
Name:       pipeline1-rq
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         0     1k
memory      0     200Gi
pods        0     10
```

   由于容器集没有运行，因此配额没有被使用。

4. 创建管道和任务。

   示例：管道的 YAML

```
apiVersion: tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: maven-build
spec:
  workspaces:
  - name: local-maven-repo
```

```
  resources:
  - name: app-git
    type: git
  tasks:
  - name: build
    taskRef:
      name: mvn
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["package"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
  - name: int-test
    taskRef:
      name: mvn
    runAfter: ["build"]
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["verify"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
  - name: gen-report
    taskRef:
      name: mvn
    runAfter: ["build"]
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["site"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
```

示例：管道中任务的 YAML

```
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: mvn
spec:
  workspaces:
  - name: maven-repo
  inputs:
```

```
    params:
    - name: GOALS
      description: The Maven goals to run
      type: array
      default: ["package"]
    resources:
    - name: source
      type: git
  steps:
    - name: mvn
      image: gcr.io/cloud-builders/mvn
      workingDir: /workspace/source
      command: ["/usr/bin/mvn"]
      args:
        - -Dmaven.repo.local=$(workspaces.maven-repo.path)
        - "$(inputs.params.GOALS)"
      priorityClassName: pipeline1-pc
```

注意

确保管道中的所有任务都属于相同的优先级类。

5. 创建并启动管道运行。

示例：管道运行的 YAML

```
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  generateName: petclinic-run-
spec:
  pipelineRef:
    name: maven-build
  resources:
  - name: app-git
    resourceSpec:
      type: git
      params:
        - name: url
          value: https://github.com/spring-projects/spring-petclinic
```

6. 创建 pod 后，验证管道运行的资源配额使用量。

示例：验证管道的资源配额使用情况

```
$ oc describe quota
```

输出示例

```
Name:       pipeline1-rq
Namespace:  default
Resource    Used  Hard
--------    ----  ----
```

```
cpu         500m  1k
memory      10Gi  200Gi
pods        1     10
```

输出表明，您可以通过指定每个优先级类的资源配额，为属于优先级类的所有并发运行 pod 管理组合资源配额。

4.12.3. 其他资源

- Kubernetes 中的资源配额
- Kubernetes 中的限制范围
- Kubernetes 中的资源请求和限值

4.13. 在特权安全上下文中使用 POD

如果 Pod 由管道运行或任务运行，则 OpenShift Pipelines 1.3.x 及更新版本的默认配置不允许使用特权安全上下文运行 Pod。对于这样的容器集，默认服务帐户为 **pipeline**，与 **pipelines** 服务帐户关联的安全性上下文约束（SCC）是 **pipelines-scc**。**pipelines-scc** SCC 与 **anyuid** SCC 类似，但存在细微差别，如管道 SCC 的 YAML 文件中定义：

pipelines-scc.yaml 片断示例

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
...
allowedCapabilities:
  - SETFCAP
...
fsGroup:
  type: MustRunAs
...
```

另外，**Buildah** 集群任务作为 OpenShift Pipelines 的一部分提供，使用 **vfs** 作为默认存储驱动程序。

4.13.1. 运行管道运行和任务运行带有特权安全上下文的 Pod

流程

要使用 **特权** 安全上下文运行 pod（从管道运行或任务运行中），请执行以下修改：

- 将关联的用户帐户或服务帐户配置为具有显式 SCC。您可以使用以下任一方法执行配置：
  - 运行以下命令：

    ```
    $ oc adm policy add-scc-to-user <scc-name> -z <service-account-name>
    ```

  - 或者，修改 **RoleBinding** 和 **Role** 或 **ClusterRole** 的 YAML 文件：

    RoleBinding 对象示例

    ```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    ```

```
metadata:
  name: service-account-name 1
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-clusterrole 2
subjects:
- kind: ServiceAccount
  name: pipeline
  namespace: default
```

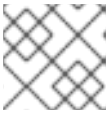
1 使用适当的服务帐户名称替换。

2 根据您使用的角色绑定，使用适当的集群角色替换。

ClusterRole 对象示例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-clusterrole 1
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use
```

1 根据您使用的角色绑定，使用适当的集群角色替换。



注意

作为最佳实践，请创建默认 YAML 文件的副本并在重复文件中进行更改。

- 如果您不使用 **vfs** 存储驱动程序，请将与任务运行或管道运行关联的服务帐户配置为具有特权 SCC，并将安全上下文设置为 **privileged: true**。

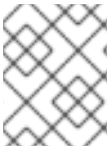
4.13.2. 使用自定义 SCC 和自定义服务帐户运行管道运行和任务

使用与默认 **pipelines** 服务帐户关联的 **pipelines-scc** 安全性上下文约束(SCC)时，管道运行和任务运行 pod 可能会面临超时问题。这是因为在默认的 **pipelines-scc** SCC 中，**fsGroup.type** 参数设置为 **MustRunAs**。

注意

有关 pod 超时的更多信息，请参阅 BZ#1995779.

为避免 pod 超时，您可以创建一个自定义 SCC，并将 **fsGroup.type** 参数设置为 **RunAsAny**，并将它与自定义服务帐户关联。



注意

作为最佳实践，使用自定义 SCC 和自定义服务帐户来运行管道运行和任务运行。这种方法具有更大的灵活性，在升级过程中修改默认值时不会中断运行。

流程

1. 定义自定义 SCC，并将 **fsGroup.type** 参数设置为 **RunAsAny** ：

示例：自定义 SCC

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: my-scc is a close replica of anyuid scc. pipelines-scc has
fsGroup - RunAsAny.
  name: my-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

2. 创建自定义 SCC：

示例：创建 my-scc SCC

```
$ oc create -f my-scc.yaml
```

3. 创建自定义服务帐户：

   示例：创建一个 fsgroup-runasany 服务帐户

   ```
   $ oc create serviceaccount fsgroup-runasany
   ```

4. 将自定义 SCC 与自定义服务帐户关联：

   示例：将 my-scc SCC 与 fsgroup-runasany 服务帐户关联

   ```
   $ oc adm policy add-scc-to-user my-scc -z fsgroup-runasany
   ```

   如果要将自定义服务帐户用于特权任务，您可以通过运行以下命令将 **privileged** SCC 与自定义服务帐户关联：

   示例：将 privileged SCC 与 fsgroup-runasany 服务帐户关联

   ```
   $ oc adm policy add-scc-to-user privileged -z fsgroup-runasany
   ```

5. 在管道运行和任务运行中使用自定义服务帐户：

   示例：Pipeline 使用 fsgroup-runasany 自定义服务帐户运行 YAML

   ```
   apiVersion: tekton.dev/v1beta1
   kind: PipelineRun
   metadata:
     name: <pipeline-run-name>
   spec:
     pipelineRef:
       name: <pipeline-cluster-task-name>
     serviceAccountName: 'fsgroup-runasany'
   ```

   示例：任务使用 fsgroup-runasany 自定义服务帐户运行 YAML

   ```
   apiVersion: tekton.dev/v1beta1
   kind: TaskRun
   metadata:
     name: <task-run-name>
   spec:
     taskRef:
       name: <cluster-task-name>
     serviceAccountName: 'fsgroup-runasany'
   ```

4.13.3. 其他资源

- 有关管理 SCC 的信息，请参阅管理安全性上下文约束。

4.14. 使用事件监听程序保护 WEBHOOK

作为管理员，您可以使用事件监听程序保护 Webhook。创建命名空间后，您可以通过将 **operator.tekton.dev/enable-annotation=enabled** 标签添加到命名空间，为 **Eventlistener** 资源启用 HTTPS。然后，您可以使用重新加密的 TLS 终止创建 **Trigger** 资源和安全路由。

Red Hat OpenShift Pipelines 中的触发器支持不安全的 HTTP 和安全 HTTPS 连接到 **Eventlistener** 资源。HTTPS 保护集群内部和外部的连接。

Red Hat OpenShift Pipelines 运行 **tekton-operator-proxy-webhook** pod，用于监视命名空间中的标签。当您将标签添加到命名空间时，webhook 在 **EventListener** 对象上设置 **service.beta.openshift.io/serving-cert-secret-name=<secret_name>** 注解。这反过来会创建 secret 和所需的证书。

```
service.beta.openshift.io/serving-cert-secret-name=<secret_name>
```

另外，您可以将创建的 secret 挂载到 **Eventlistener** pod 中，以保护请求。

4.14.1. 提供与 OpenShift 路由的安全连接

要使用重新加密的 TLS 终止创建路由，请运行：

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=<hostname>
```

或者，您可以创建一个重新加密的 TLS 终止 YAML 文件来创建安全路由。

重新加密 TLS 终止 YAML 示例，以创建安全路由

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend (2)
  tls:
    termination: reencrypt (3)
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- (4)
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

(1) (2) 对象的名称，仅限于 63 个字符。

(3) termination 字段设置为 **reencrypt**。这是唯一需要的 TLS 字段。

(4) 这是重新加密所必需的。**destinationCACertificate** 字段指定一个 CA 证书来验证端点证书，从而保护从路由器到目的地 Pod 的连接。您可以在以下情况之一中省略此字段：

- 服务使用服务签名证书。

- 管理员为路由器指定默认 CA 证书，服务具有由该 CA 签名的证书。

您可以运行 **oc create route reencrypt --help** 命令显示更多选项。

4.14.2. 使用安全 HTTPS 连接创建示例 EventListener 资源

本节使用 pipelines-tutorial 示例来演示使用安全 HTTPS 连接创建示例 EventListener 资源。

流程

1. 从 pipelines-tutorial 存储库中的 YAML 文件创建 **TriggerBinding** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-
tutorial/master/03_triggers/01_binding.yaml
```

2. 从 pipelines-tutorial 存储库中的 YAML 文件创建 **TriggerTemplate** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-
tutorial/master/03_triggers/02_template.yaml
```

3. 直接从 pipelines-tutorial 存储库创建 **Trigger** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-
tutorial/master/03_triggers/03_trigger.yaml
```

4. 使用安全 HTTPS 连接创建 **EventListener** 资源：

   a. 添加一个标签，在 Eventlistener 资源中启用安全 **HTTPS** 连接：

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

   b. 从 pipelines-tutorial 存储库中的 YAML 文件创建 **EventListener** 资源：

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-
tutorial/master/03_triggers/04_event_listener.yaml
```

   c. 创建带有重新加密 TLS 终止的路由：

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-
cert=ca.crt --hostname=<hostname>
```

4.15. 使用 GIT SECRET 验证管道

Git 机密由凭据组成，可以安全地与 Git 存储库交互，通常用于自动执行身份验证。在 Red Hat OpenShift Pipelines 中，您可以使用 Git secret 验证管道运行和在执行过程中与 Git 存储库交互的任务运行。

管道运行或任务运行通过关联的服务帐户获取对 secret 的访问权限。管道支持将 Git secret 用作基于基本身份验证和基于 SSH 的身份验证的注解（密钥值对）。

4.15.1. 凭证选择

管道运行或任务运行可能需要多次身份验证才能访问不同的 Git 存储库。使用 Pipelines 可以使用其凭证的域注解每个 secret。

Git secret 的凭证注解键必须以 **tekton.dev/git-** 开头，其值是您要管道使用该凭证的主机的 URL。

在以下示例中，Pipelines 使用 **basic-auth** secret（依赖于用户名和密码）访问位于 **github.com** 和 **gitlab.com** 的存储库。

示例：用于基本身份验证的多个凭证

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: github.com
    tekton.dev/git-1: gitlab.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> 1
  password: <password> 2
```

1 软件仓库的用户名

2 存储库的密码或个人访问令牌

您还可以使用 **ssh-auth** secret（私钥）来访问 Git 存储库。

示例：用于基于 SSH 的身份验证的私钥

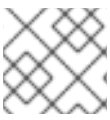
```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 1
```

1 SSH 私钥文件的内容。

4.15.2. 为 Git 配置基本身份验证

管道若要从密码保护的存储库检索资源，您必须为该管道配置基本身份验证。

要为管道配置基本身份验证，请使用指定存储库的 Git secret 中的凭证更新 **secret.yaml**、**serviceaccount.yaml** 和 **run.yaml** 文件。完成此过程后，Pipelines 可使用该信息来检索指定的管道资源。



注意

对于 GitHub，已弃用使用普通密码进行身份验证。而应使用个人访问令牌。

流程

1. 在 **secret.yaml** 文件中，指定用户名和密码或 GitHub 个人访问令牌来访问目标 Git 存储库。

```
apiVersion: v1
kind: Secret
metadata:
  name: basic-user-pass (1)
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> (2)
  password: <password> (3)
```

(1) secret 的名称。在本例中，**basic-user-pass**。

(2) Git 存储库的用户名。

(3) Git 存储库的密码。

2. 在 **serviceaccount.yaml** 文件中，将 secret 与适当的服务帐户关联。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot (1)
secrets:
  - name: basic-user-pass (2)
```

(1) 服务帐户的名称。在本例中，**build-bot**。

(2) secret 的名称。在本例中，**basic-user-pass**。

3. 在 **run.yaml** 文件中，将服务帐户与任务运行或管道运行关联。

- 将服务帐户与任务运行关联：

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 (1)
spec:
  serviceAccountName: build-bot (2)
  taskRef:
    name: build-push (3)
```

(1) 任务运行的名称。在本例中，**build-push-task-run-2**。

(2) 服务帐户的名称。在本例中，**build-bot**。

(3) 任务的名称。在本例中，**build-push**。

- 将服务帐户与 **PipelineRun** 资源关联：

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline 1
  namespace: default
spec:
  serviceAccountName: build-bot 2
  pipelineRef:
    name: demo-pipeline 3
```

1. 管道运行的名称。在本例中，**demo-pipeline**。
2. 服务帐户的名称。在本例中，**build-bot**。
3. 管道的名称。在本例中，**demo-pipeline**。

4. 应用更改。

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.15.3. 为 Git 配置 SSH 身份验证

若要让管道从配置了 SSH 密钥的存储库检索资源，您必须为该管道配置基于 SSH 的身份验证。

要为管道配置基于 SSH 的身份验证，请使用指定存储库的 SSH 私钥中的凭证更新 **secret.yaml**、**serviceaccount.yaml** 和 **run.yaml** 文件。完成此过程后，Pipelines 可使用该信息来检索指定的管道资源。

注意

考虑使用基于 SSH 的身份验证而不是基本身份验证。

流程

1. 生成 SSH 私钥，或复制通常在 **~/.ssh/id_rsa** 文件中提供的现有私钥。
2. 在 **secret.yaml** 文件中，将 **ssh-privatekey** 的值设置为 SSH 私钥文件的内容，并将 **known_hosts** 的值设置为已知主机文件的内容。

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-key 1
  annotations:
    tekton.dev/git-0: github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 2
  known_hosts: 3
```

1. 包含 SSH 私钥的机密的名称。在本例中，**ssh-key**。
2. SSH 私钥文件的内容。
3. 已知主机文件的内容。

小心

如果省略私钥，Pipelines 接受任何服务器的公钥。

3. 可选：要指定一个自定义 SSH 端口，请在 **annotation** 值的末尾添加 **:<port number>**.例如：**tekton.dev/git-0: github.com:2222**。
4. 在 **serviceaccount.yaml** 文件中，将 **ssh-key** secret 与 **build-bot** 服务帐户关联。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot (1)
secrets:
  - name: ssh-key (2)
```

1. 服务帐户的名称。在本例中，**build-bot**。
2. 包含 SSH 私钥的机密的名称。在本例中，**ssh-key**。

5. 在 **run.yaml** 文件中，将服务帐户与任务运行或管道运行关联。
   - 将服务帐户与任务运行关联：

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 (1)
spec:
  serviceAccountName: build-bot (2)
  taskRef:
    name: build-push (3)
```

1. 任务运行的名称。在本例中，**build-push-task-run-2**。
2. 服务帐户的名称。在本例中，**build-bot**。
3. 任务的名称。在本例中，**build-push**。

   - 将服务帐户与管道运行关联：

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline (1)
  namespace: default
```

```
spec:
  serviceAccountName: build-bot 2
  pipelineRef:
    name: demo-pipeline 3
```

1. 管道运行的名称。在本例中，**demo-pipeline**。
2. 服务帐户的名称。在本例中，**build-bot**。
3. 管道的名称。在本例中，**demo-pipeline**。

6. 应用更改。

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.15.4. 在 git 类型任务中使用 SSH 身份验证

在调用 Git 命令时，您可以在任务的步骤中直接使用 SSH 身份验证。SSH 身份验证忽略 **$HOME** 变量，并且仅使用 **/etc/passwd** 文件中指定的用户主目录。因此，任务中的每个步骤都必须将 **/tekton/home/.ssh** 目录符号链接到相关用户的主目录。

但是，当您使用 **git** 类型的管道资源或 Tekton 目录中提供的 **git-clone** 任务时，不需要显式符号链接。

有关在 **git** 类型任务中使用 SSH 身份验证的示例，请参阅 authenticating-git-commands.yaml。

4.15.5. 以非 root 用户身份使用 secret

在某些情况下，您可能需要将 secret 用作非 root 用户，例如：

- 容器用于执行运行的用户和组由平台随机化。
- 任务中的步骤定义非 root 安全性上下文。
- 任务指定一个全局非 root 安全上下文，它应用到任务中的所有步骤。

在这种情况下，请考虑以非 root 用户身份运行任务和管道运行的以下方面：

- Git 的 SSH 身份验证要求用户在 **/etc/passwd** 目录中配置有效的主目录。指定没有有效主目录的 UID 会导致身份验证失败。
- SSH 身份验证会忽略 **$HOME** 环境变量。因此，您必须将由 Pipelines（**/tekton/home**） 定义的 **$HOME** 目录中的 secret 文件符号链接到非 root 用户的有效主目录。

此外，若要在非 root 安全上下文中配置 SSH 身份验证，请参阅对 git 命令进行身份验证的示例。

4.15.6. 限制对特定步骤的 secret 访问

默认情况下，Pipelines 的 secret 存储在 **$HOME/tekton/home** 目录中，并可用于任务中的所有步骤。

要将 secret 限制为特定的步骤，请使用 secret 定义指定卷，并在特定步骤中挂载卷。

4.16. 为 OPENSHIFT PIPELINES 提供链安全使用 TEKTON 链

重要

Tekton 链只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

Tekton Chains 是一个 Kubernetes 自定义资源定义(CRD)控制器。您可以使用它来管理使用 Red Hat OpenShift Pipelines 创建的任务和管道的供应链安全。

默认情况下，Tekton Chains 会观察 OpenShift Container Platform 集群中的所有任务运行执行。当任务运行完成时，Tekton Chains 会获取任务运行的快照。然后，它会将快照转换为一个或多个标准有效负载格式，最后签署并存储所有工件。

要捕获有关任务运行的信息，Tekton Chains 使用 **Result** 和 **PipelineResource** 对象。当对象不可用时，Tekton 会链 OCI 镜像的 URL 和合格摘要。

注意

PipelineResource 对象已弃用，并将在以后的发行版本中删除；对于手动使用，建议使用 **Results** 对象。

4.16.1. 主要特性

- 您可以使用加密密钥类型和服务（如 **cosign** ）为任务运行、任务运行结果和 OCI registry 镜像签名。
- 您可以使用"测试"格式，如 **in-toto**。
- 您可以使用 OCI 存储库作为存储后端安全存储签名和签名工件。

4.16.2. 使用 Red Hat OpenShift Pipelines Operator 安装 Tekton 链

集群管理员可以使用 **TektonChain** 自定义资源(CR)来安装和管理 Tekton 链。

注意

Tekton Chains 是 Red Hat OpenShift Pipelines 的一个可选组件。目前，您无法使用 **TektonConfig** CR 安装它。

先决条件

- 确保在集群中的 **openshift-pipelines** 命名空间中安装了 Red Hat OpenShift Pipelines Operator。

流程

1. 为 OpenShift Container Platform 集群创建 **TektonChain** CR。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonChain
metadata:
```

```
  name: chain
spec:
  targetNamespace: openshift-pipelines
```

2. 应用 **TektonChain** CR。

```
$ oc apply -f TektonChain.yaml 1
```

1 使用 **TektonChain** CR 的文件名替换。

3. 检查安装的状态。

```
$ oc get tektonchains.operator.tekton.dev
```

4.16.3. 配置 Tekton 链

Tekton Chains 在 **openshift-pipelines** 命名空间中使用名为 **chains-config** 的 **ConfigMap** 对象进行配置。

要配置 Tekton 链，请使用以下示例：

示例：配置 Tekton 链

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":{"artifacts.oci.storage": "",
"artifacts.taskrun.format":"tekton", "artifacts.taskrun.storage": "tekton"}}' 1
```

1 在 JSON 有效负载中使用支持的键值对组合。

4.16.3.1. Tekton Chains 配置支持的键

集群管理员可以使用各种支持的键和值来配置任务运行、OCI 镜像和存储的规格。

4.16.3.1.1. 任务运行支持的键

表 4.13. 链配置：任务运行支持的密钥

支持的键	描述	支持的值	默认值
artifacts.taskrun.format	存储任务运行有效负载的格式。	**tekton**, **in-toto**	**tekton**
artifacts.taskrun.storage	任务运行签名的存储后端。您可以将多个后端指定为用逗号分开的列表，如 **"tekton,oci"**。要禁用此工件，请提供一个空字符串 **""**。	**tekton**, **oci**	**tekton**
artifacts.taskrun.signer	签名后端为任务运行有效负载进行签名。	**x509**	**x509**

4.16.3.1.2. OCI 支持的密钥

表 4.14. 链配置： OCI 支持的密钥

支持的键	描述	支持的值	默认值
artifacts.oci.format	存储 OCI 有效负载的格式。	**simplesigning**	**simplesigning**
artifacts.oci.storage	OCI 签名的存储后端。您可以将多个后端指定为用逗号分开的列表，如 **"oci,tekton"**。要禁用 OCI 工件，请提供空字符串 **""**。	**tekton, oci**	**oci**
artifacts.oci.signer	签名后端以签署 OCI 有效负载。	**x509,cosign**	**x509**

4.16.3.1.3. 支持的存储密钥

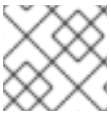
表 4.15. 链配置：存储支持的密钥

支持的键	描述	支持的值	默认值
artifacts.oci.repository	用于存储 OCI 签名的 OCI 存储库。	目前，链仅支持内部 OpenShift OCI registry；不支持 Quay 等其他流行选项。	

4.16.4. 在 Tekton Chains 中签名 secret

集群管理员可以生成密钥对，并使用 Tekton 链来使用 Kubernetes secret 为工件签名。要使 Tekton 链正常工作，加密的密钥和密码必须作为 **openshift-pipelines** 命名空间中的 **signing-secrets** Kubernetes secret 的一部分存在。

目前，Tekton 链支持 **x509** 和 **cosign** 签名方案。



注意

只使用一个受支持的签名方案。

4.16.4.1. 使用 x509 进行签名

要将 **x509** 签名方案与 Tekton Chains 搭配使用，请将 **ed25519** 或 **ecdsa** 类型的 **x509.pem** 私钥存储在 **signing-secrets** Kubernetes secret 中。确保密钥保存为未加密的 PKCS8 PEM 文件(**BEGIN PRIVATE KEY**)。

4.16.4.2. 使用 cosign 进行签名

使用 Tekton 链的 **cosign** 签名方案：

1. 安装 cosign。
2. 生成 **cosign.key** 和 **cosign.pub** 密钥对。

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

   Cosign 提示您输入密码，并创建一个 Kubernetes secret。

3. 将加密的 **cosign.key** 私钥和 **cosign.password** 解密密码存储在 **signing-secrets** Kubernetes secret 中。确保私钥存储为 **ENCRYPTED COSIGN PRIVATE KEY** 类型的加密 PEM 文件。

4.16.4.3. 签名故障排除

如果签名 secret 已填充，您可能会遇到以下错误：

```
Error from server (AlreadyExists): secrets "signing-secrets" already exists
```

要解决这个问题：

1. 删除 secret：

```
$ oc delete secret signing-secrets -n openshift-pipelines
```

2. 重新创建密钥对并使用您首选的签名方案将其存储在 secret 中。

4.16.5. 对 OCI registry 进行身份验证

在将签名推送到 OCI Registry 之前，集群管理员必须配置 Tekton 链，以便与 registry 进行身份验证。Tekton Chains 控制器使用与任务运行相同的服务帐户。要设置具有所需凭证(push)到 OCI registry 的服务帐户，请执行以下步骤：

流程

1. 设置 Kubernetes 服务帐户的命名空间和名称。

```
$ export NAMESPACE=<namespace> 1
$ export SERVICE_ACCOUNT_NAME=<service_account> 2
```

   1 与服务帐户关联的命名空间。

   2 服务帐户的名称。

2. 创建 Kubernetes secret。

```
$ oc create secret registry-credentials \
  --from-file=.dockerconfigjson \ 1
  --type=kubernetes.io/dockerconfigjson \
  -n $NAMESPACE
```

   1 使用 Docker 配置文件的路径替换。默认路径为 **~/.docker/config.json**。

3. 授予服务帐户对 secret 的访问权限。

```
$ oc patch serviceaccount $SERVICE_ACCOUNT_NAME \
  -p "{\"imagePullSecrets\": [{\"name\": \"registry-credentials\"}]}" -n $NAMESPACE
```

如果对 Red Hat OpenShift Pipelines 分配到所有任务的默认 **pipeline** 服务帐户进行补丁，Red Hat OpenShift Pipelines Operator 将覆盖服务帐户。作为最佳实践，您可以执行以下步骤：

a. 创建单独的服务帐户，以分配给用户的任务运行。

```
$ oc create serviceaccount <service_account_name>
```

b. 通过设置任务运行模板中的 **serviceaccountname** 字段的值，将服务帐户关联到运行任务。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
name: build-push-task-run-2
spec:
serviceAccountName: build-bot (1)
taskRef:
  name: build-push
...
```

1 使用新创建的服务帐户的名称替换。

4.16.5.1. 创建和验证任务运行签名而无需任何其他身份验证

要验证使用 Tekton 链与任何其他身份验证一起运行的任务的签名，请执行以下任务：

- 创建加密的 x509 密钥对，并将它保存为 Kubernetes secret。
- 配置 Tekton Chains 后端存储。
- 创建任务运行，为它签名并将签名和有效负载存储为任务运行自身时的注解。
- 从已签名任务运行中检索签名和有效负载。
- 验证任务运行的签名。

先决条件

确保在集群中安装了以下内容：

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- Cosign

流程

1. 创建加密的 x509 密钥对，并将它保存为 Kubernetes secret：

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

提示时提供密码。Cosign 将生成的私钥存储为 **openshift-pipelines** 命名空间中的 **signing-secrets** Kubernetes secret 的一部分。

2. 在 Tekton Chains 配置中，禁用 OCI 存储，并将任务运行存储和格式设置为 **tekton**。

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":{"artifacts.oci.storage":
"", "artifacts.taskrun.format":"tekton", "artifacts.taskrun.storage": "tekton"}}'
```

3. 重启 Tekton Chains 控制器，以确保应用了修改后的配置。

```
$ oc delete po -n openshift-pipelines -l app=tekton-chains-controller
```

4. 创建任务运行。

```
$ oc create -f
https://raw.githubusercontent.com/tektoncd/chains/main/examples/taskruns/task-output-
image.yaml (1)
taskrun.tekton.dev/build-push-run-output-image-qbjvh created
```

(1) 使用指向您的任务运行的 URI 或文件路径替换。

5. 检查步骤的状态，并等待 till 进程完成。

```
$ tkn tr describe --last
[...truncated output...]
NAME                          STATUS
· create-dir-builtimage-9467f   Completed
· git-source-sourcerepo-p2sk8   Completed
· build-and-push                Completed
· echo                        Completed
· image-digest-exporter-xlkn7   Completed
```

6. 从存储为 **base64** 编码注解的对象检索签名和有效负载：

```
$ export TASKRUN_UID=$(tkn tr describe --last -o  jsonpath='{.metadata.uid}')
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\.tekton\.dev/signature-
taskrun-$TASKRUN_UID}" > signature
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\.tekton\.dev/payload-
taskrun-$TASKRUN_UID}" | base64 -d > payload
```

7. 验证签名。

```
$ cosign verify-blob --key k8s://openshift-pipelines/signing-secrets --signature ./signature
./payload
Verified OK
```

4.16.6. 使用 Tekton 链来签名和验证镜像并证明

集群管理员可以通过执行以下任务来使用 Tekton 链来签名和验证镜像和验证镜像：

- 创建加密的 x509 密钥对，并将它保存为 Kubernetes secret。
- 为 OCI registry 设置身份验证，以在测试过程中存储镜像、镜像签名和签名的镜像。

- 配置 Tekton 链以生成和签署认可。
- 在任务运行中，使用 Kaniko 创建镜像。
- 验证已签名的镜像及已签名证明。

先决条件

确保在集群中安装了以下内容：

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- Cosign
- Rekor
- jq

流程

1. 创建加密的 x509 密钥对，并将它保存为 Kubernetes secret：

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

提示时提供密码。Cosign 将生成的私钥作为 openshift-pipelines 命名空间中的 **signing-secrets** Kubernetes secret 的一部分存储在 **openshift-pipelines** 命名空间中，并将公钥写入 **cosign.pub** 本地文件。

2. 为镜像 registry 配置身份验证。

   a. 要将 Tekton Chains 控制器配置为将签名推送到 OCI registry，请使用与任务运行服务帐户关联的凭证。如需更多信息，请参阅"授权到 OCI registry"部分。

   b. 要为构建并推送到 registry 的 Kaniko 任务配置身份验证，请创建一个包含所需凭证的 docker **config.json** 文件的 Kubernetes secret。

```
$ oc create secret generic <docker_config_secret_name> \ 1
  --from-file <path_to_config.json> 2
```

   1 使用 docker config secret 的名称替换。

   2 使用 docker **config.json** 文件的路径替换。

3. 通过在 **chains-config** 对象中设置 **artifacts.taskrun.format**、**artifacts.taskrun.storage** 和 **transparency.enabled** 参数来配置 Tekton 链 ：

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.format": "in-toto"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.storage": "oci"}}'
```

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"transparency.enabled": "true"}}'
```

4. 启动 Kaniko 任务。

   a. 将 Kaniko 任务应用到集群。

   ```
   $ oc apply -f examples/kaniko/kaniko.yaml 1
   ```

   1 使用 Kaniko 任务的 URI 或文件路径替换。

   b. 设置适当的环境变量。

   ```
   $ export REGISTRY=<url_of_registry> 1

   $ export DOCKERCONFIG_SECRET_NAME=
   <name_of_the_secret_in_docker_config_json> 2
   ```

   1 使用您要推送镜像的 registry 的 URL 替换。

   2 使用 docker **config.json** 文件中的 secret 名称替换。

   c. 启动 Kaniko 任务。

   ```
   $ tkn task start --param IMAGE=$REGISTRY/kaniko-chains --use-param-defaults --
   workspace name=source,emptyDir="" --workspace
   name=dockerconfig,secret=$DOCKERCONFIG_SECRET_NAME kaniko-chains
   ```

   观察此任务的日志，直到所有步骤都完成。身份验证成功后，最终镜像将推送到 **$REGISTRY/kaniko-chains**。

5. 等待一分钟，以允许 Tekton 链生成证据并对其进行签名，然后在任务运行时检查 **chains.tekton.dev/signed=true** 注解的可用性。

   ```
   $ oc get tr <task_run_name> \ 1
   -o json | jq -r .metadata.annotations

   {
     "chains.tekton.dev/signed": "true",
     ...
   }
   ```

   1 使用任务运行的名称替换。

6. 验证镜像和 attestation。

   ```
   $ cosign verify --key cosign.pub $REGISTRY/kaniko-chains

   $ cosign verify-attestation --key cosign.pub $REGISTRY/kaniko-chains
   ```

7. 在 Rekor 中找到镜像的验证情况。

a. 获取 $REGISTRY/kaniko-chains 镜像摘要。您可以搜索任务运行或拉取镜像以提取摘要。

b. 搜索 Rekor 以查找与镜像 **sha256** 摘要匹配的所有条目。

```
$ rekor-cli search --sha <image_digest> 1

<uuid_1> 2
<uuid_2> 3
...
```

1. 使用镜像的 **sha256** 摘要替换。
2. 第一个匹配通用唯一标识符(UUID)。
3. 第二个匹配 UUID。

搜索结果显示匹配条目的 UUID。其中其中一个 UUID 包含 attestation。

c. 检查 attestation。

```
$ rekor-cli get --uuid <uuid> --format json | jq -r .Attestation | base64 --decode | jq
```

4.16.7. 其他资源

- 安装 OpenShift Pipelines

4.17. 使用 OPENSHIFT LOGGING OPERATOR 查看管道日志

管道运行、任务运行和事件侦听器生成的日志存储在其各自的 pod 中。检查和分析用于故障排除和审计的日志非常有用。

但是，保留 pod 不会造成不必要的资源消耗和杂乱的命名空间。

要消除对 pod 查看管道日志的依赖，您可以使用 OpenShift Elasticsearch Operator 和 OpenShift Logging Operator。这些 Operator 可帮助您使用 Elasticsearch Kibana 堆栈查看管道日志，即使您删除了包含日志的 pod。

4.17.1. 先决条件

在 Kibana 仪表板中尝试查看管道日志前，请确保以下内容：

- 步骤由集群管理员执行。
- 管道运行和任务运行的日志可用。
- 安装了 OpenShift Elasticsearch Operator 和 OpenShift Logging Operator。

4.17.2. 在 Kibana 中查看管道日志

在 Kibana web 控制台中查看管道日志：

流程

1. 以集群管理员身份登录到 OpenShift Container Platform Web 控制台。
2. 在菜单栏右上角，点击 **grid** 图标 → **Observability** → **Logging**。这时会显示 Kibana Web 控制台。
3. 创建索引模式：
   a. 在 **Kibana** Web 控制台左侧导航面板中，点击 **Management**。
   b. 单击 **Create index pattern**。
   c. 在 **Step 1 of 2: Define index pattern**→ **Index pattern** 中输入一个 * 特征并点 **Next Step**。
   d. 在 **Step 2 of 2: Configure settings**→ **Time filter field name** 中，从下来菜单中选择 **@timestamp**，点 **Create index pattern**。
4. 添加过滤器：
   a. 在 **Kibana** Web 控制台左侧导航面板中，点 **Discover**。
   b. 点 **Add a filter +**→ **Edit Query DSL**。

注意

- 对于以下每个示例过滤器，编辑查询并单击 **Save**。
- 这些过滤器会逐个应用。

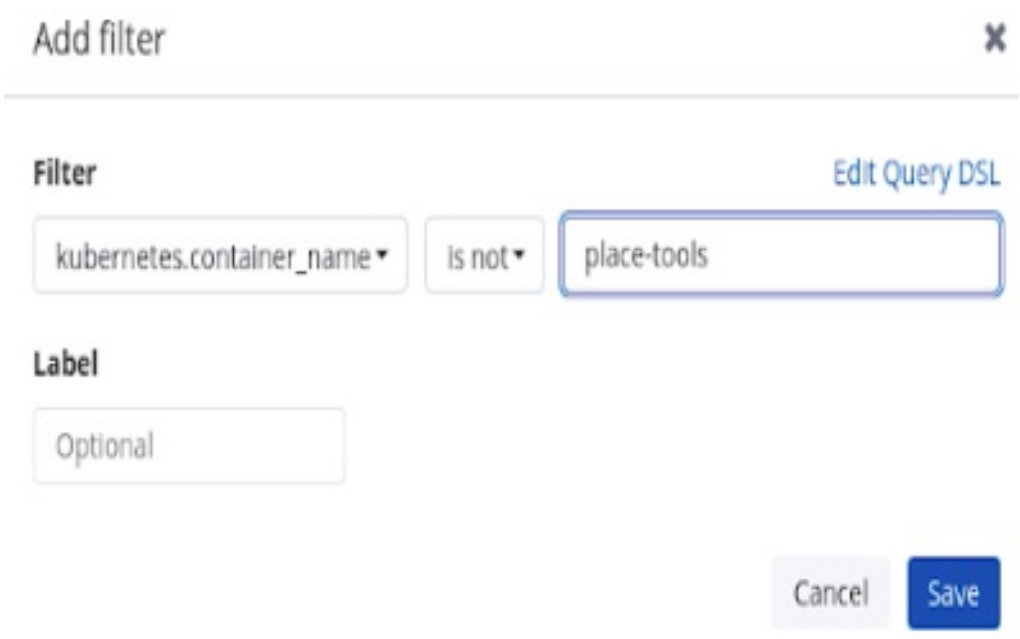
i. 过滤与管道相关的容器：

过滤管道容器的查询示例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "app_kubernetes_io/managed-by=tekton-pipelines",
     "type": "phrase"
   }
 }
  }
}
```

ii. 过滤所有不是 **place-tools** 容器的容器。作为使用图形下拉菜单而不是编辑查询 DSL 的一个示例，请考虑以下方法：

图 4.6. 使用下拉列表字段进行过滤示例



iii. 在标签中过滤 **pipelinerun** 以高亮显示：

在标签中过滤 pipelinerun 的查询示例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "tekton_dev/pipelineRun=",
     "type": "phrase"
   }
 }
  }
}
```

iv. 在标签中过滤 **pipeline** 以高亮显示：

在标签中过滤 pipeline 以高亮显示的查询示例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "tekton_dev/pipeline=",
     "type": "phrase"
   }
 }
  }
}
```

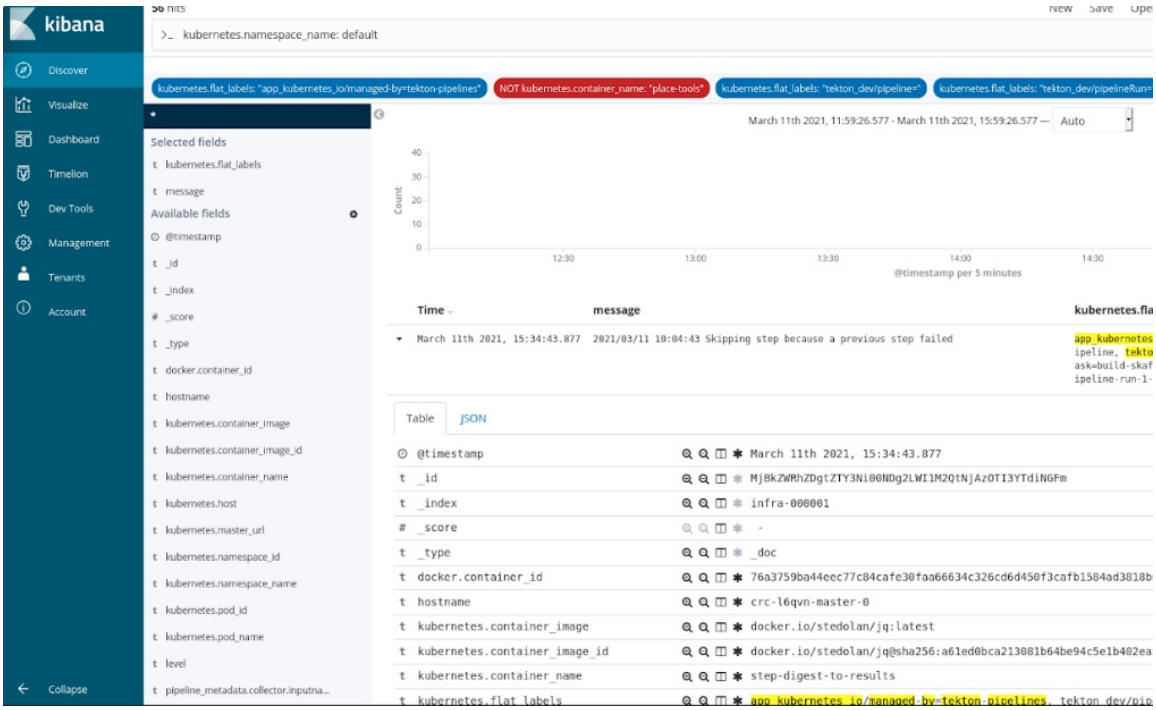
c. 从 **Available fields** 列表中，选择以下字段：

- **kubernetes.flat_labels**

- **message**
  确保所选字段显示在 **Selected fields** 列表下。

d. 日志显示在 **message** 字段下。

图 4.7. 过滤的消息



4.17.3. 其他资源

- 安装 OpenShift Logging
- 查看资源的日志
- 使用 Kibana 查看集群日志

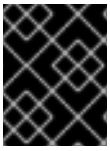
4.18. 以非 ROOT 用户身份使用 BUILDAH 构建容器镜像

作为 root 用户在容器上运行 Pipelines，可以将容器进程和主机公开给其他潜在的恶意资源。您可以作为容器中的特定非 root 用户运行工作负载来降低此类风险。要以非 root 用户身份使用 Buildah 运行容器镜像的构建，可以执行以下步骤：

- 定义自定义服务帐户 (SA) 和安全性上下文约束 (SCC)。
- 配置 Buildah，以使用 ID 为 **1000** 的 **build** 用户。
- 使用自定义配置映射启动任务运行，或将其与管道运行集成。

4.18.1. 配置自定义服务帐户和安全上下文约束

默认 **pipeline** SA 允许使用命名空间范围之外的用户 ID。要减少对默认 SA 的依赖性，您可以为 ID 为 **1000** 的 **build** 用户定义具有必要的集群角色和角色绑定的自定义 SA 和 SCC。



重要

目前，Buildah 需要启用 **allowPrivilegeEscalation** 设置，才能在容器中运行。通过这个设置，Buildah 可以在以非 root 用户身份运行时利用 **SETUID** 和 **SETGID** 功能。

流程

- 使用必要的集群角色和角色绑定创建自定义 SA 和 SCC。

  示例：用户 id 为 1000 的自定义 SA 和 SCC。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-sa-userid-1000 1
---
kind: SecurityContextConstraints
metadata:
  annotations:
  name: pipelines-scc-userid-1000 2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true 3
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
- KILL
runAsUser: 4
  type: MustRunAs
  uid: 1000
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-userid-1000-clusterrole 5
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - pipelines-scc-userid-1000
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pipelines-scc-userid-1000-rolebinding 6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-userid-1000-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-sa-userid-1000
```

1. 定义一个自定义 SA。
2. 使用修改的 **runAsUser** 字段，定义基于受限特权创建的自定义 SCC。
3. 目前，Buildah 需要启用 **allowPrivilegeEscalation** 设置，才能在容器中运行。通过这个设置，Buildah 可以在以非 root 用户身份运行时利用 **SETUID** 和 **SETGID** 功能。
4. 限制通过自定义 SA 附加了自定义 SCC 的 Pod，使其以用户 ID **1000** 身份运行。
5. 定义使用自定义 SCC 的集群角色。
6. 将使用自定义 SCC 的集群角色绑定到自定义 SA。

4.18.2. 配置 Buildah 以使用 build 用户

您可以定义一个 Buildah 任务，以使用带有用户 ID **1000** 的 **build** 用户。

流程

1. 作为普通任务，创建 **buildah** 集群任务的副本。

```
$ oc get clustertask buildah -o yaml | yq '. |= (del .metadata |= with_entries(select(.key == "name" )))' | yq '.kind="Task"' | yq '.metadata.name="buildah-as-user"' | oc create -f -
```

2. 编辑复制的 **buildah** 任务。

```
$ oc edit task buildah-as-user
```

示例：使用 build 用户修改 Buildah 任务

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: buildah-as-user
spec:
  description: >-
    Buildah task builds source into a container image and
    then pushes it to a container registry.
    Buildah Task builds source into a container image using Project Atomic's
    Buildah build tool.It uses Buildah's support for building from Dockerfiles,
    using its buildah bud command.This command executes the directives in the
    Dockerfile to assemble a container image, then pushes that image to a
    container registry.
  params:
  - name: IMAGE
    description: Reference of the image buildah will produce.
  - name: BUILDER_IMAGE
    description: The location of the buildah builder image.
    default:
registry.redhat.io/rhel8/buildah@sha256:99cae35f40c7ec050fed3765b2b27e0b8bbea2aa2da7
c16408e2ca13c60ff8ee
  - name: STORAGE_DRIVER
    description: Set buildah storage driver
    default: vfs
  - name: DOCKERFILE
    description: Path to the Dockerfile to build.
    default: ./Dockerfile
  - name: CONTEXT
    description: Path to the directory to use as context.
    default: .
  - name: TLSVERIFY
    description: Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)
    default: "true"
  - name: FORMAT
    description: The format of the built container, oci or docker
    default: "oci"
  - name: BUILD_EXTRA_ARGS
    description: Extra parameters passed for the build command when building images.
    default: ""
  - description: Extra parameters passed for the push command when pushing images.
    name: PUSH_EXTRA_ARGS
    type: string
    default: ""
  - description: Skip pushing the built image
    name: SKIP_PUSH
    type: string
    default: "false"
  results:
  - description: Digest of the image just built.
    name: IMAGE_DIGEST
    type: string
  workspaces:
  - name: source
  steps:
```

```
  - name: build
    securityContext:
      runAsUser: 1000 (1)
    image: $(params.BUILDER_IMAGE)
    workingDir: $(workspaces.source.path)
    script: |
      echo "Running as USER ID `id`" (2)
      buildah --storage-driver=$(params.STORAGE_DRIVER) bud \
        $(params.BUILD_EXTRA_ARGS) --format=$(params.FORMAT) \
        --tls-verify=$(params.TLSVERIFY) --no-cache \
        -f $(params.DOCKERFILE) -t $(params.IMAGE) $(params.CONTEXT)
      [[ "$(params.SKIP_PUSH)" == "true" ]] && echo "Push skipped" && exit 0
      buildah --storage-driver=$(params.STORAGE_DRIVER) push \
        $(params.PUSH_EXTRA_ARGS) --tls-verify=$(params.TLSVERIFY) \
        --digestfile $(workspaces.source.path)/image-digest $(params.IMAGE) \
        docker://$(params.IMAGE)
      cat $(workspaces.source.path)/image-digest | tee /tekton/results/IMAGE_DIGEST
    volumeMounts:
    - name: varlibcontainers
      mountPath: /home/build/.local/share/containers (3)
  volumes:
  - name: varlibcontainers
    emptyDir: {}
```

(1) 以用户 id **1000** 明确运行容器，它与 Buildah 镜像中的 **build** 用户对应。

(2) 显示用户 id，以确认进程以用户 ID **1000** 身份运行。

(3) 您可以根据需要更改卷挂载的路径。

4.18.3. 使用自定义配置映射或管道运行启动任务运行

定义自定义 Buildah 集群任务后，您可以创建一个 **TaskRun** 对象，该对象以 **build** 用户，使用用户 id **1000** 来构建镜像。另外，您还可以将 **TaskRun** 对象集成为 **PipelineRun** 对象的一部分。

流程

1. 使用自定义 **ConfigMap** 和 **Dockerfile** 对象创建一个 **TaskRun** 对象。

   示例：以用户 ID 1000 身份运行 Buildah 的任务运行

```
apiVersion: v1
data:
  Dockerfile: |
    ARG BASE_IMG=registry.access.redhat.com/ubi8/ubi
    FROM $BASE_IMG AS buildah-runner
    RUN dnf -y update && \
        dnf -y install git && \
        dnf clean all
    CMD git
kind: ConfigMap
metadata:
  name: dockerfile (1)
---
```

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: buildah-as-user-1000
spec:
  serviceAccountName: pipelines-sa-userid-1000 2
  params:
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  taskRef:
    kind: Task
    name: buildah-as-user
  workspaces:
  - configMap:
      name: dockerfile 3
    name: source
```

1 使用配置映射，因为重点是在任务运行中，而无需使用 Dockerfile 获取某些源的以前的任务。

2 您创建的服务帐户的名称。

3 将配置映射挂载为 **buildah-as-user** 任务的源工作区。

2. （可选）创建管道和对应的管道运行。

示例：管道和对应的管道运行

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-buildah-as-user-1000
spec:
  params:
  - name: IMAGE
  - name: URL
  workspaces:
  - name: shared-workspace
  - name: sslcertdir
    optional: true
  tasks:
  - name: fetch-repository 1
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.URL)
    - name: subdirectory
      value: ""
    - name: deleteExisting
```

```
      value: "true"
  - name: buildah
    taskRef:
      name: buildah-as-user 2
    runAfter:
    - fetch-repository
    workspaces:
    - name: source
      workspace: shared-workspace
    - name: sslcertdir
      workspace: sslcertdir
    params:
    - name: IMAGE
      value: $(params.IMAGE)
---
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: pipelinerun-buildah-as-user-1000
spec:
  taskRunSpecs:
    - pipelineTaskName: buildah
      taskServiceAccountName: pipelines-sa-userid-1000 3
  params:
  - name: URL
    value: https://github.com/openshift/pipelines-vote-api
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  pipelineRef:
    name: pipeline-buildah-as-user-1000
  workspaces:
  - name: shared-workspace 4
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 100Mi
```

1. 使用 **git-clone** 集群任务获取包含 Dockerfile 的源，并使用修改后的 Buildah 任务构建它。
2. 请参阅修改后的 Buildah 任务。
3. 使用您为 Buildah 任务创建的服务帐户。
4. 使用控制器自动创建的持久性卷声明 (PVC) 在 **git-clone** 任务和修改后的 Buildah 任务间共享数据。

3. 启动任务运行或管道运行。

4.18.4. 无特权构建的限制

无特权构建的进程可用于大多数 **Dockerfile** 对象。但是，一些已知的限制可能会导致构建失败：

- 由于缺少必要权限问题，使用 **--mount=type=cache** 选项可能会失败。如需更多信息，请参阅本文档。
- 使用 **--mount=type=secret** 选项会失败，因为挂载资源需要未由自定义 SCC 提供的额外功能。

其他资源

- 管理安全性上下文约束 (SCC)

第 5 章 GITOPS

5.1. RED HAT OPENSHIFT GITOPS 发行注记

Red Hat OpenShift GitOps 是为云原生应用程序实施持续部署的一种声明方法。当应用程序部署到不同环境中的不同集群时，Red Hat OpenShift GitOps 可确保应用程序的一致性，如开发、临时和生产环境。Red Hat OpenShift GitOps 可帮助您自动执行以下任务：

- 确保集群具有类似的配置、监控和存储状态
- 从已知状态恢复或重新创建集群
- 对多个 OpenShift Container Platform 集群应用或恢复配置更改
- 将模板配置与不同环境关联
- 在集群间（从调试到生产阶段）推广应用程序。

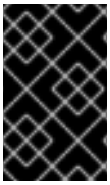
如需了解 Red Hat OpenShift GitOps 的概述，请参阅了解 OpenShift GitOps。

5.1.1. 兼容性和支持列表

这个版本中的一些功能当前还只是一个技术预览。它们并不适用于在生产环境中使用。

在下表中，被标记为以下状态的功能：

- **TP**: *技术预览*
- **GA**: *正式发行*
- **NA**:*不适用*



重要

在 OpenShift Container Platform 4.13 中，**stable** 频道已被删除。在升级到 OpenShift Container Platform 4.13 之前，如果您已在 **stable** 频道中，请选择适当的频道并切换到它。

OpenShift GitOps	组件版本							OpenShift 版本
版本	**kam**	**Helm**	**Kustomize**	**Argo CD**	**ApplicationSet**	**Dex**	**RH SSO**	
1.8.0	0.0.47 TP	3.10.0 GA	4.5.7 GA	2.6.3 GA	不适用	2.35.1 GA	7.5.1 GA	4.10-4.13
1.7.0	0.0.46 TP	3.10.0 GA	4.5.7 GA	2.5.4 GA	不适用	2.35.1 GA	7.5.1 GA	4.10-4.12

OpenShift GitOps	组件版本							OpenShift 版本
1.6.0	0.0.46 TP	3.8.1 GA	4.4.1 GA	2.4.5 GA	GA 并包含在 ArgoCD 组件中	2.30.3 GA	7.5.1 GA	4.8-4.11
1.5.0	0.0.42 TP	3.8.0 GA	4.4.1 GA	2.3.3 GA	0.4.1 TP	2.30.3 GA	7.5.1 GA	4.8-4.11
1.4.0	0.0.41 TP	3.7.1 GA	4.2.0 GA	2.2.2 GA	0.2.0 TP	2.30.0 GA	7.4.0 GA	4.7-4.10
1.3.0	0.0.40 TP	3.6.0 GA	4.2.0 GA	2.1.2 GA	0.2.0 TP	2.28.0 GA	7.4.0 GA	4.7-4.9, 4.6 带有有限的 GA 支持
1.2.0	0.0.38 TP	3.5.0 GA	3.9.4 GA	2.0.5 GA	0.1.0 TP	不适用	7.4.0 GA	4.8
1.1.0	0.0.32 TP	3.5.0 GA	3.9.4 GA	2.0.0 GA	不适用	不适用	不适用	4.7

- **kam** 是 Red Hat OpenShift GitOps Application Manager 命令行界面 (CLI)。
- RH SSO 是 Red Hat SSO 的缩写。

5.1.1.1. 技术预览功能

下表中提到的功能当前还只是一个技术预览 (TP)。它们并不适用于在生产环境中使用。

表 5.1. 技术预览

功能	Red Hat OpenShift GitOps 版本中的技术预览（TP）	Red Hat OpenShift GitOps 版本中的正式版本（GA）
ApplicationSet Progressive Rollout 策略	1.8.0	不适用
一个应用程序的多个源	1.8.0	不适用
非 control plane 命名空间中的 Argo CD 应用程序	1.7.0	不适用
Argo CD 通知控制器	1.6.0	不适用

功能	Red Hat OpenShift GitOps 版本中的技术预览（TP）	Red Hat OpenShift GitOps 版本中的正式版本（GA）
OpenShift Container Platform Web 控制台的 **Developer** 视角中的 Red Hat OpenShift GitOps **Environments** 页面	1.1.0	不适用

5.1.2. 使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

5.1.3. Red Hat OpenShift GitOps 1.8.4 发行注记

Red Hat OpenShift GitOps 1.8.4 现在包括在 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 中。

5.1.3.1. 新功能

当前发行版本包括以下改进：

- 在这个版本中，捆绑的 Argo CD 更新至 2.6.13 版本。

5.1.3.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当命名空间和应用程序增加时，Argo CD 可能会变得无响应。竞争资源的功能会导致死锁。在这个版本中，通过删除死锁解决了这个问题。现在，当命名空间或应用程序增加时，您不会遇到崩溃或无响应的问题。GITOPS-3192

- 在此次更新之前，当重新同步应用程序时，Argo CD 应用程序控制器资源可能会突然停止工作。在这个版本中解决了这个问题，方法是添加逻辑以防止集群缓存死锁。现在，应用程序应该可以成功重新同步。GITOPS-3052

- 在此次更新之前，在 **argocd-ssh-known-hosts-cm** 配置映射中已知主机的 RSA 密钥中存在不匹配。在这个版本中，通过将 RSA 密钥与上游项目匹配解决了这个问题。现在，您可以在默认部署中使用默认 RSA 密钥。GITOPS-3144

- 在此次更新之前，部署 Red Hat OpenShift GitOps Operator 时会使用一个旧的 Redis 镜像版本，这会导致漏洞。在这个版本中，通过将 Redis 升级到 **registry.redhat.io/rhel-8/redis-6** 镜像的最新版本解决了 Redis 中的漏洞。GITOPS-3069

- 在此次更新之前，用户无法通过 Operator 部署的 Argo CD 连接到 Microsoft Team Foundation Server (TFS) 类型 Git 存储库。在这个版本中，通过将 Git 版本更新至 Operator 中的 2.39.3 解决了这个问题。现在，您可以在存储库配置过程中设置 **Force HTTP basic auth** 标志，以与 TFS 类型 Git 存储库连接。GITOPS-1315

5.1.3.3. 已知问题

- 目前，Red Hat OpenShift GitOps 1.8.4 不适用于 OpenShift Container Platform 4.10 和 4.11 的 **latest** 频道。**latest** 频道由 GitOps 1.9.z 获取，它只在 OpenShift Container Platform 4.12 及更新的版本中发布。
  作为临时解决方案，切换到 **gitops-1.8** 频道来获取新的更新。GITOPS-3158

5.1.4. Red Hat OpenShift GitOps 1.8.3 发行注记

Red Hat OpenShift GitOps 1.8.3 现在包括在 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 中。

5.1.4.1. 勘误更新

5.1.4.1.1. RHBA-2023:3206 和 RHSA-2023:3229 - Red Hat OpenShift GitOps 1.8.3 安全更新公告

发布日期：2023 年 5 月 18 日

此发行版本中包括的安全修复列表包括在以下公告中：

- RHBA-2023:3206
- RHSA-2023:3229

如果安装了 Red Hat OpenShift GitOps Operator，请运行以下命令来查看此发行版本中的容器镜像：

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.4.2. 修复的问题

- 在此次更新之前，当启用 **Autoscale** 且 pod 横向自动扩展 (HPA) 控制器试图编辑服务器部署中的副本设置时，Operator 会覆盖它。另外，在自动扩展参数指定的任何更改都会被正确传播到集群中的 HPA。在这个版本中解决了这个问题。现在，只有在禁用 **Autoscale** 且正确更新 HPA 参数时，Operator 才会在副本偏移上协调。GITOPS-2629

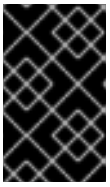
5.1.5. Red Hat OpenShift GitOps 1.8.2 发行注记

Red Hat OpenShift GitOps 1.8.2 现在包括在 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 中。

5.1.5.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当使用 **.spec.dex** 参数配置 Dex 时，并尝试使用 **LOG IN VIA OPENSHIFT** 选项登录到 Argo CD UI，您将无法登录。在这个版本中解决了这个问题。



重要

ArgoCD CR 中的 **spec.dex** 参数已弃用。在以后的 Red Hat OpenShift GitOps v1.9 发行版本中，计划使用 ArgoCD CR 中的 **spec.dex** 参数配置 Dex。考虑改用 **.spec.sso** 参数。请参阅 "使用 .spec.sso 启用或禁用 Dex"。 GITOPS-2761

- 在此次更新之前，集群和 **kam** CLI pod 无法在 OpenShift Container Platform 4.10 集群上进行新的 Red Hat OpenShift GitOps v1.8.0 安装。在这个版本中解决了这个问题，现在所有 pod 都会如预期运行。GITOPS-2762

5.1.6. Red Hat OpenShift GitOps 1.8.1 发行注记

Red Hat OpenShift GitOps 1.8.1 现在包括在 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 中。

5.1.6.1. 勘误更新

5.1.6.1.1. RHSA-2023:1452 - Red Hat OpenShift GitOps 1.8.1 安全更新公告

发布日期：2023 年 3 月 23 日

此发行版本中包括的安全修复列表包括在 RHSA-2023:1452 公告中。

如果安装了 Red Hat OpenShift GitOps Operator，请运行以下命令来查看此发行版本中的容器镜像：

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.7. Red Hat OpenShift GitOps 1.8.0 发行注记

Red Hat OpenShift GitOps 1.8.0 现在包括在 OpenShift Container Platform 4.10、4.11、4.12 和 4.13 中。

5.1.7.1. 新功能

当前发行版本包括以下改进：

- 在这个版本中，您可以添加 ApplicationSet Progressive Rollout 策略功能的支持。使用此功能，您可以在修改 ApplicationSet spec 或 Application 模板后，增强 ArgoCD ApplicationSet 资源，以便为进度应用程序资源更新嵌入推出部署策略。当您启用此功能时，应用程序会以声明性顺序更新，而不是同时更新。GITOPS-956

重要

ApplicationSet Progressive Rollout 策略是一个技术预览功能。

- 在这个版本中，OpenShift Container Platform Web 控制台的 **Developer** 视角中的 **Application environments** 页面与 Red Hat OpenShift GitOps Application Manager 命令行界面 (CLI) **kam** 分离。您不必使用 **kam** CLI 为环境生成应用程序环境清单，以便在 OpenShift Container Platform Web 控制台的 **Developer** 视角中显示。您可以使用自己的清单，但环境仍必须由命名空间表示。另外，仍然需要特定的标签和注解。GITOPS-1785

- 在这个版本中，Red Hat OpenShift GitOps Operator 和 **kam** CLI 可用于 OpenShift Container Platform 上的 ARM 架构。GITOPS-1688

重要

spec.sso.provider: keycloak 在 ARM 上还不被支持。

- 在这个版本中，您可以通过将 **.spec.monitoring.enabled** 标志值设置为 **true** 来为特定 Argo CD 实例启用工作负载监控。因此，Operator 会创建一个 **PrometheusRule** 对象，其中包含每个 Argo CD 组件的警报规则。当在一定的时间段内，相应组件的副本数偏离了期望的状态，则这些警报规则会触发警报。Operator 不会覆盖用户对 **PrometheusRule** 对象所做的更改。GITOPS-2459

- 在这个版本中，您可以使用 Argo CD CR 将命令参数传递给存储库服务器部署。GITOPS-2445

例如：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  repo:
    extraRepoCommandArgs:
      - --max.combined.directory.manifests.size
      - 10M
```

5.1.7.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，您只能在 **openshift-gitops-repo-server** pod 中设置 **ARGOCD_GIT_MODULES_ENABLED** 环境变量，而不是在 **ApplicationSet Controller** pod 中设置。因此，在使用 Git 生成器时，Git 子模块会在生成子应用程序时克隆，因为 **ApplicationSet Controller** 环境中缺少变量。另外，如果在 ArgoCD 中没有配置克隆这些子模块所需的凭证，应用程序生成会失败。在这个版本中解决了这个问题，您可以使用 Argo CD CR 将任何环境变量（如 **ArgoCD_GIT_MODULES_ENABLED**）添加到 **ApplicationSet Controller** pod 中。然后 **ApplicationSet Controller** pod 从克隆的存储库中成功生成子应用程序，且进程中没有克隆子模块。GITOPS-2399
  例如：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  applicationSet:
    env:
     - name: ARGOCD_GIT_MODULES_ENABLED
       value: "true"
```

- 在此次更新之前，在安装 Red Hat OpenShift GitOps Operator v1.7.0 时，为身份验证 Dex 创建的默认 **argocd-cm.yml** 配置映射文件包含 base64 编码的客户端 secret，格式为 **key:value** 对。在这个版本中解决了这个问题，将客户端 secret 存储在默认的 **argocd-cm.yml** 配置映射文件中。现在，客户端 secret 位于 **argocd-secret** 对象中，您可以在配置映射中将其引用为 secret 名称。GITOPS-2570

5.1.7.3. 已知问题

- 当您在不使用 **kam** CLI 的情况下使用清单部署应用程序，并在 OpenShift Container Platform Web 控制台的 **Developer** 视角中的 **Application environments** 页面中查看应用程序，则相应应用程序的 Argo CD URL 不会从卡中的 Argo CD 图标按预期加载页面。GITOPS-2736

5.1.8. Red Hat OpenShift GitOps 1.7.4 发行注记

Red Hat OpenShift GitOps 1.7.4 现在包括在 OpenShift Container Platform 4.10、4.11 和 4.12 中。

5.1.8.1. 勘误更新

5.1.8.1.1. RHSA-2023:1454 - Red Hat OpenShift GitOps 1.7.4 安全更新公告

发布日期：2023 年 3 月 23 日

此发行版本中包括的安全修复列表包括在 RHSA-2023:1454 公告中。

如果安装了 Red Hat OpenShift GitOps Operator，请运行以下命令来查看此发行版本中的容器镜像：

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.9. Red Hat OpenShift GitOps 1.7.3 发行注记

Red Hat OpenShift GitOps 1.7.3 现在包括在 OpenShift Container Platform 4.10、4.11 和 4.12 中。

5.1.9.1. 勘误更新

5.1.9.1.1. RHSA-2023:1454 - Red Hat OpenShift GitOps 1.7.3 安全更新公告

发布日期：2023 年 3 月 23 日

此发行版本中包括的安全修复列表包括在 RHSA-2023:1454 公告中。

如果安装了 Red Hat OpenShift GitOps Operator，请运行以下命令来查看此发行版本中的容器镜像：

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.10. Red Hat OpenShift GitOps 1.7.1 发行注记

Red Hat OpenShift GitOps 1.7.1 现在包括在 OpenShift Container Platform 4.10、4.11 和 4.12 中。

5.1.10.1. 勘误更新

5.1.10.1.1. RHSA-2023:0467 - Red Hat OpenShift GitOps 1.7.1 安全更新公告

发布日期：2023 年 1 月 25 日

此发行版本中包括的安全修复列表包括在 RHSA-2023:0467 公告中。

如果安装了 Red Hat OpenShift GitOps Operator，请运行以下命令来查看此发行版本中的容器镜像：

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.11. Red Hat OpenShift GitOps 1.7.0 发行注记

Red Hat OpenShift GitOps 1.7.0 现在包括在 OpenShift Container Platform 4.10、4.11 和 4.12 中。

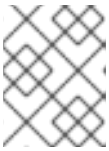
5.1.11.1. 新功能

当前发行版本包括以下改进：

- 在这个版本中，您可以在 Notifications 控制器中添加环境变量。GITOPS-2313
- 在这个版本中，默认的 nodeSelector **"kubernetes.io/os": "linux"** 键值对添加到所有工作负载中，以便它们只在 Linux 节点上调度。另外，任何自定义节点选择器都会添加到默认值，如果它们具有相同的键，则优先选择它们。GITOPS-2215
- 在这个版本中，您可以通过编辑 **GitopsService** 自定义资源来在 Operator 工作负载中设置自定义节点选择器。GITOPS-2164
- 在这个版本中，您可以使用 RBAC 策略匹配器模式从以下选项中选择： **glob** （默认）和 **regex**.GITOPS-1975
- 在这个版本中，您可以使用以下附加子键自定义资源行为：

Subkey	键形式	argocd-cm 中映射的字段
resourceHealthChecks	resource.customizations.health.<group_kind>	resource.customizations.health
resourceIgnoreDifferences	resource.customizations.ignoreDifferences.<group_kind>	resource.customizations.ignoreDifferences
resourceActions	resource.customizations.actions.<group_kind>	resource.customizations.actions

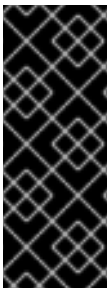
GITOPS-1561



注意

在以后的发行版本中，可以通过仅使用 resourceCustomization 而不是 subkeys 弃用旧方法来自定义资源行为。

- 在这个版本中，要使用 **Developer** 视角中的 **Environments** 页面，如果您使用 Red Hat OpenShift GitOps 1.7 之前的版本以及 OpenShift Container Platform 4.15 或更高版本时，需要进行升级。GITOPS-2415
- 在这个版本中，您可以在同一集群中的任何命名空间中创建由同一 control plane Argo CD 实例管理的应用程序。作为管理员，执行以下操作以启用此更新：
  - 将命名空间添加到管理应用程序的集群范围的 Argo CD 实例的 **.spec.sourceNamespaces** 属性中。
  - 将命名空间添加到与应用程序关联的 **AppProject** 自定义资源中的 **.spec.sourceNamespaces** 属性中。
    GITOPS-2341



重要

非 control plane 命名空间中的 Argo CD 应用程序只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，Argo CD 支持 Server-Side Apply 功能，这有助于用户执行以下任务：
  - 管理对于允许注解大小为 262144 字节的大型资源。
  - 对未由 Argo CD 管理或部署的现有资源进行补丁。
    您可以在应用程序或资源级别配置此功能。GITOPS-2340

5.1.11.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当 **anyuid** SCC 分配给 Dex 服务帐户时会导致 Dex pod 失败并带有 **CreateContainerConfigError** 错误，这会影响到 Red Hat OpenShift GitOp 的发布。在这个版本中，通过将默认用户 ID 分配给 Dex 容器解决了这个问题。GITOPS-2235
- 在此次更新之前，Red Hat OpenShift GitOps 在 Dex 之外通过 OIDC 使用 RHSSO (Keycloak)。但是，当使用未由知名证书颁发机构签名的证书配置时，当 RHSSO 证书配置时，无法验证 RHSSO 证书。在这个版本中解决了这个问题，您可以提供一个自定义证书，以便在与它通信时验证 KeyCloak 的 TLS 证书。另外，您可以将 **rootCA** 添加到 Argo CD 自定义资源 **.spec.keycloak.rootCA** 字段中。Operator 使用 PEM 编码的 root 证书协调此更改并更新 **oidc.config in argocd-cm** 配置映射。GITOPS-2214

使用 Keycloak 配置的 Argo CD 示例：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  sso:
    keycloak:
      rootCA: '<PEM encoded root certificate>'
    provider: keycloak
.......
.......
```

- 在此次更新之前，应用程序控制器会因为存活度探测的无响应而多次重启。在这个版本中，通过删除 **statefulset** 应用控制器中的存活度探测解决了这个问题。GITOPS-2153

5.1.11.3. 已知问题

- 在此次更新之前，Operator 不会协调仓库服务器 **的 mountatoken** 和 **ServiceAccount** 设置。虽然这个问题已被解决，删除服务帐户不会恢复到默认值。GITOPS-1873
- 临时解决方案：手动将 **spec.repo.serviceaccountfield 设置为 default** 服务帐户。GITOPS-2452

5.1.12. Red Hat OpenShift GitOps 1.6.7 发行注记

Red Hat OpenShift GitOps 1.6.7 现在包括在 OpenShift Container Platform 4.8, 4.9, 4.10, 和 4.11 中。

5.1.12.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，从 v0.5.0 开始的所有版本的 Argo CD Operator 都容易受到信息泄漏漏洞的影响。因此，未授权的用户可以通过检查 API 错误消息并使用发现的应用程序名称作为另一个攻击的起点来枚举应用程序名称。例如，攻击者可能会利用他们了解应用程序名称来让管理员授予更高的特权。在这个版本中解决了 CVE-2022-41354 错误。GITOPS-2635, CVE-2022-41354

5.1.13. Red Hat OpenShift GitOps 1.6.6 发行注记

Red Hat OpenShift GitOps 1.6.6 现在包括在 OpenShift Container Platform 4.8, 4.9, 4.10, 和 4.11 中。

5.1.13.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，从 v0.5.0 开始的所有版本的 Argo CD Operator 都容易受到信息泄漏漏洞的影响。因此，未授权的用户可以通过检查 API 错误消息并使用发现的应用程序名称作为另一个攻击的起点来枚举应用程序名称。例如，攻击者可能会利用他们了解应用程序名称来让管理员授予更高的特权。在这个版本中解决了 CVE-2022-41354 错误。GITOPS-2635, CVE-2022-41354

5.1.14. Red Hat OpenShift GitOps 1.6.4 发行注记

Red Hat OpenShift GitOps 1.6.4 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.14.1. 修复的问题

- 在此次更新之前，Argo CD v1.8.2 及之后的版本的所有版本都会受到不正确的授权错误的影响。因此，Argo CD 可以接受可能不适用于访问集群的受众的令牌。这个问题现已解决。CVE-2023-22482

5.1.15. Red Hat OpenShift GitOps 1.6.2 发行注记

Red Hat OpenShift GitOps 1.6.2 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 上。

5.1.15.1. 新功能

- 此发行版本从 **openshift-gitops-operator** CSV 文件中删除 **DISABLE_DEX** 环境变量。因此，在执行全新的 Red Hat OpenShift GitOps 安装时，不再设置此环境变量。GITOPS-2360

5.1.15.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当在一个项目中安装了超过 5 个 Operator 时，订阅健康检查会为缺失的 **InstallPlan** 标记为 **degraded**。在这个版本中解决了这个问题。GITOPS-2018

- 在此次更新之前，Red Hat OpenShift GitOps Operator 会在检测到 Argo CD 实例使用已弃用的字段时，使用弃用通知警告来垃圾邮件集群。在这个版本中解决了这个问题，为每个检测到字段的实例只显示一个警告事件。GITOPS-2230

- 在 OpenShift Container Platform 4.12 中，安装控制台是可选的。在这个版本中，更新了 Red Hat OpenShift GitOps Operator，以防止安装控制台时出现 Operator 错误。GITOPS-2352

5.1.16. Red Hat OpenShift GitOps 1.6.1 发现注记

Red Hat OpenShift GitOps 1.6.1 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.16.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，因为存活度探测的无响应，在应用程序控制器的大型应用程序中会多次重启。在这个版本中，通过删除应用程序控制器 **StatefulSet** 对象中的存活度探测解决了这个问题。GITOPS-2153

- 在此次更新之前，当使用不受证书颁发机构签名的证书设置时，将无法验证 RHSSO 证书。在这个版本中解决了这个问题，您可以提供一个自定义证书，它会在与它通信时验证 Keycloak 的 TLS 证书。您可以将 **rootCA** 添加到 Argo CD 自定义资源 **.spec.keycloak.rootCA** 字段中。Operator 使用 PEM 编码的 root 证书协调此更改并更新 **argocd-cm ConfigMap** 中的 **oidc.config** 字段。GITOPS-2214

  注意

  REST在更新 **.spec.keycloak.rootCA** 字段后重启 Argo CD 服务器 pod。

  例如：

  ```
  apiVersion: argoproj.io/v1alpha1
  kind: ArgoCD
  metadata:
    name: example-argocd
    labels:
      example: basic
  spec:
    sso:
      provider: keycloak
      keycloak:
       rootCA: |
         ---- BEGIN CERTIFICATE ----
         This is a dummy certificate
         Please place this section with appropriate rootCA
         ---- END CERTIFICATE ----
    server:
      route:
        enabled: true
  ```

- 在此次更新之前，由 Argo CD 管理的终止命名空间会阻止创建角色和其他受管命名空间配置。在这个版本中解决了这个问题。GITOPS-2277

- 在此次更新之前，当将 **anyuid** 的 SCC 分配给 Dex **ServiceAccount** 资源时，Dex Pod 无法启动 **CreateContainerConfigError**。在这个版本中，通过将默认用户 ID 分配给 Dex 容器解决了这个问题。GITOPS-2235

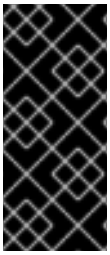
5.1.17. Release notes for Red Hat OpenShift GitOps 1.6.0

Red Hat OpenShift GitOps 1.6.0 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.17.1. 新功能

当前发行版本包括以下改进：

- 在以前的版本中，Argo CD **ApplicationSet** 控制器是一个技术预览 (TP) 功能。在这个版本中，它是一个正式发行 (GA) 功能。GITOPS-1958
- 在这个版本中，Red Hat OpenShift GitOps 的最新版本包括在 **latest** 和基于版本的频道中。要获取这些升级，请更新 **Subscription** 对象 YAML 文件中的 **channel** 参数：将其值从 **stable** 改为 **latest** 或基于版本的频道，如 **gitops-1.6**。GITOPS-1791
- 在这个版本中，控制 keycloak 配置的 **spec.sso** 字段的参数将移到 **.spec.sso.keycloak** 中。**.spec.dex** 字段的参数已添加到 **.spec.sso.dex** 中。使用 **.spec.sso.provider** 开始启用或禁用 Dex。**.spec.dex** 参数已弃用，计划在版本 1.9 中删除，以及 keycloak 配置的 **DISABLE_DEX** 和 **.spec.sso** 字段。GITOPS-1983
- 在这个版本中，Argo CD Notifications 控制器作为可选的工作负载使用，可以使用 Argo CD 自定义资源中的 **.spec.notifications.enabled** 参数启用或禁用。Argo CD Notifications 控制器作为技术预览提供。GITOPS-1917



重要

Argo CD Notifications 控制器只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

- 在这个版本中，Tekton 管道运行的资源排除，默认情况下会添加任务运行。Argo CD，默认修剪这些资源。这些资源排除会添加到从 OpenShift Container Platform 创建的新的 Argo CD 实例中。如果通过 CLI 创建实例，则不添加这些资源。GITOPS-1876
- 在这个版本中，您可以通过在 Operand 的规格中设置 **resourceTrackingMethod** 参数来选择 Argo CD 使用的跟踪方法。GITOPS-1862
- 在这个版本中，您可以使用 Red Hat OpenShift GitOps Argo CD 自定义资源的 **extraConfig** 字段在 **argocd-cm** configMap 中添加条目。指定的条目在没有验证的情况下被协调到实时 **config-cm** configMap 中。GITOPS-1964
- 在这个版本中，在 OpenShift Container Platform 4.11 中，**Developer** 视角中的 Red Hat OpenShift GitOps **Environments** 页面会显示应用程序环境成功部署的历史记录，以及指向每个部署的修订版本的链接。GITOPS-1269
- 在这个版本中，您可以使用 Argo CD 管理资源，这些资源也被 Operator 用作模板资源或 "source"。GITOPS-982
- 在这个版本中，Operator 会使用正确的权限配置 Argo CD 工作负载，以满足为 Kubernetes 1.24 启用的 Pod Security Admission。GITOPS-2026
- 在这个版本中，支持配置管理插件 2.0。您可以使用 Argo CD 自定义资源为仓库服务器指定边栏容器。GITOPS-776
- 在这个版本中，Argo CD 组件和 Redis 缓存之间的所有通信都使用现代 TLS 加密进行安全保护。GITOPS-720
- 此 Red Hat OpenShift GitOps 发行版本添加了对 OpenShift Container Platform 4.10 的 IBM Z 和 IBM Power 的支持。目前，IBM Z 和 IBM Power 不支持在受限环境中安装。

5.1.17.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，**system:serviceaccount:argocd:gitops-argocd-application-controller** 无法在命名空间 **webapps-dev** 的 API 组 **monitoring.coreos.com** 中创建资源 "prometheusrules"。在这个版本中解决了这个问题，Red Hat OpenShift GitOps 现在可以从 **monitoring.coreos.com** API 组中管理所有资源。GITOPS-1638

- 在此次更新之前，在协调集群权限时，如果 secret 属于集群配置实例，则它已被删除。在这个版本中解决了这个问题。现在，secret 中的 **namespaces** 字段已被删除，而不是 secret。GITOPS-1777

- 在此次更新之前，如果您通过 Operator 安装 Argo CD 的 HA 变体，Operator 会创建带有 **podAffinity** 规则的 Redis **StatefulSet** 对象，而不是 **podAntiAffinity** 规则。在这个版本中解决了这个问题，Operator 会根据 **podAntiAffinity** 规则创建 Redis **StatefulSet** 规则。GITOPS-1645

- 在此次更新之前，Argo CD **ApplicationSet** 具有太多 **ssh** Zombie 进程。在这个版本中解决了这个问题：它为 **ApplicationSet** 控制器添加了 tini，它是一个生成进程并获取 zombies 的简单 init 守护进程。这可确保 **SIGTERM** 信号正确传递给正在运行的进程，防止它成为 Zombie 进程。GITOPS-2108

5.1.17.3. 已知问题

- Red Hat OpenShift GitOps Operator 可以在 Dex 之外通过 OIDC 使用 RHSSO (KeyCloak)。但是，在应用了最新的安全修复程序时，在某些情况下无法验证 RHSSO 证书。GITOPS-2214 作为临时解决方案，请在 ArgoCD 规格中禁用 OIDC (Keycloak/RHSSO) 端点的 TLS 验证。

```
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
...
```

5.1.18. Red Hat OpenShift GitOps 1.5.9 发行注记

Red Hat OpenShift GitOps 1.5.9 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.18.1. 修复的问题

- 在此次更新之前，Argo CD v1.8.2 及之后的版本的所有版本都会受到不正确的授权错误的影响。因此，Argo CD 可以接受可能无法获得访问集群的用户的令牌。这个问题现已解决。CVE-2023-22482

5.1.19. Red Hat OpenShift GitOps 1.5.7 发行注记

Red Hat OpenShift GitOps 1.5.7 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.19.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在 OpenShift Container Platform 4.12 中，安装控制台是可选的。在这个版本中，更新了 Red Hat OpenShift GitOps Operator，以防止安装控制台时出现 Operator 错误。GITOPS-2353

5.1.20. Red Hat OpenShift GitOps 1.5.6 发现注记

Red Hat OpenShift GitOps 1.5.6 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.20.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，因为存活度探测的无响应，在应用程序控制器的大型应用程序中会多次重启。在这个版本中，通过删除应用程序控制器 **StatefulSet** 对象中的存活度探测解决了这个问题。GITOPS-2153

- 在此次更新之前，当使用不受证书颁发机构签名的证书设置时，将无法验证 RHSSO 证书。在这个版本中解决了这个问题，您可以提供一个自定义证书，它会在与它通信时验证 Keycloak 的 TLS 证书。您可以将 **rootCA** 添加到 Argo CD 自定义资源 **.spec.keycloak.rootCA** 字段中。Operator 使用 PEM 编码的 root 证书协调此更改并更新 **argocd-cm ConfigMap** 中的 **oidc.config** 字段。GITOPS-2214

  注意

  REST在更新 **.spec.keycloak.rootCA** 字段后重启 Argo CD 服务器 pod。

  例如：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
     rootCA: |
       ---- BEGIN CERTIFICATE ----
       This is a dummy certificate
       Please place this section with appropriate rootCA
       ---- END CERTIFICATE ----
  server:
    route:
      enabled: true
```

- 在此次更新之前，由 Argo CD 管理的终止命名空间会阻止创建角色和其他受管命名空间配置。在这个版本中解决了这个问题。GITOPS-2278

- 在此次更新之前，当将 **anyuid** 的 SCC 分配给 Dex **ServiceAccount** 资源时，Dex Pod 无法启动 **CreateContainerConfigError**。在这个版本中，通过将默认用户 ID 分配给 Dex 容器解决了这个问题。GITOPS-2235

5.1.21. Red Hat OpenShift GitOps 1.5.5 的发行注记

Red Hat OpenShift GitOps 1.5.5 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.21.1. 新功能

当前发行版本包括以下改进：

- 在这个版本中，捆绑的 Argo CD 更新至 2.3.7 版本。

5.1.21.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当集群中存在更严格的 SCC 时，ArgoCD 实例的 **redis-ha-haproxy** Pod 会失败。在这个版本中，通过更新工作负载中的安全上下文解决了这个问题。GITOPS-2034

5.1.21.3. 已知问题

- Red Hat OpenShift GitOps Operator 可以使用带有 OIDC 和 Dex 的 RHSSO (KeyCloak)。但是，在应用了最新的安全修复程序时，Operator 无法验证 RHSSO 证书。GITOPS-2214
  作为临时解决方案，请在 ArgoCD 规格中禁用 OIDC (Keycloak/RHSSO) 端点的 TLS 验证。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
...
```

5.1.22. Red Hat OpenShift GitOps 1.5.4 发行注记

Red Hat OpenShift GitOps 1.5.4 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.22.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，Red Hat OpenShift GitOps 使用一个较老版本的 **REDIS 5** 镜像标签。在这个版本中解决了这个问题，并升级 **rhel8/redis-5** 镜像标签。GITOPS-2037

5.1.23. Red Hat OpenShift GitOps 1.5.3 发行注记

Red Hat OpenShift GitOps 1.5.3 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.23.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，Argo CD v1.0.0 及之后的版本的所有未修补版本都会受到跨站点脚本错误的影响。因此，未授权的用户无法在 UI 中注入 javascript 链接。这个问题现已解决。CVE-2022-31035

- 在此次更新之前，当从 Argo CD CLI 或 UI 启动 SSO 登录时，所有版本的 Argo CD v0.11.0 及更新版本都容易受到多个攻击的影响。这个问题现已解决。CVE-2022-31034

- 在此次更新之前，Argo CD v0.7 及更新版本的未修补版本都会容易受到内存消耗错误的影响。因此，未授权用户可以使 Argo CD 的 repo-server 崩溃。这个问题现已解决。CVE-2022-31016

- 在此次更新之前，Argo CD v1.3.0 及之后的版本的所有未修补版本都会受到符号链接跟踪错误的影响。因此，具有存储库写入访问权限的未授权用户可以泄漏 Argo CD 的 repo-server 中的敏感 YAML 文件。这个问题现已解决。CVE-2022-31036

5.1.24. Red Hat OpenShift GitOps 1.5.2 发行注记

Red Hat OpenShift GitOps 1.5.2 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.24.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，缺少 **redhat-operator-index** 引用的镜像。这个问题现已解决。GITOPS-2036

5.1.25. Red Hat OpenShift GitOps 1.5.1 发行注记

Red Hat OpenShift GitOps 1.5.1 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.25.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，如果启用了 Argo CD 的匿名访问，则未经身份验证的用户可以制作一个 JWT 令牌，并获得 Argo CD 实例的完整访问权限。这个问题现已解决。CVE-2022-29165
- 在此次更新之前，未经身份验证的用户可以在启用 SSO 时在登录屏幕上显示错误消息。这个问题现已解决。CVE-2022-24905
- 在此次更新之前，Argo CD v0.7.0 及更新版本的未修补版本会受到符号链接跟踪程序错误修复的影响。因此，具有存储库写入访问权限的未授权用户可以泄漏 Argo CD 的 repo-server 中的敏感文件。这个问题现已解决。CVE-2022-24904

5.1.26. Red Hat OpenShift GitOps 1.5.0 发行注记

Red Hat OpenShift GitOps 1.5.0 现在包括在 OpenShift Container Platform 4.8、4.9、4.10 和 4.11 中。

5.1.26.1. 新功能

当前发行版本包括以下改进：

- 此功能增强将 Argo CD 升级到 2.3.3 版本。GITOPS-1708
- 此功能增强将 Dex 升级到 2.30.3 版本。GITOPS-1850
- 在这个版本中，将 Helm 升级到 3.8.0。GITOPS-1709
- 在这个版本中，将 Kustomize 升级到 4.4.1 版本。GITOPS-1710
- 此功能增强将应用程序设置为版本 0.4.1。
- 在这个版本中，添加了名称 **latest** 的新频道，它提供 Red Hat OpenShift GitOps 的最新版本。对于 GitOps v1.5.0，Operator 被推送到 **gitops-1.5**、**latest** 频道和现有的 **stable** 频道。从 GitOps v1.6 中，所有最新版本只会推送到 **latest** 频道，而不是 **stable** 频道。GITOPS-1791

- 在这个版本中，新 CSV 添加 **olm.skipRange: '>=1.0.0 <1.5.0'** 注解。因此，所有之前的版本都会被跳过。Operator 直接升级到 v1.5.0。GITOPS-1787
- 在这个版本中，Operator 将 Red Hat Single Sign-On(RH-SSO)更新至 v7.5.1 版本，包括以下改进：
  - 您可以使用包含 **kube:admin** 凭证的 OpenShift 凭证登录到 Argo CD。
  - RH-SSO 支持并使用 OpenShift 组为基于角色的访问控制(RBAC)配置 Argo CD 实例。
  - RH-SSO 遵循 **HTTP_Proxy** 环境变量。您可以使用 RH-SSO 作为在代理后面的 Argo CD 的 SSO。
    GITOPS-1330
- 在这个版本中，一个新的 **.host** URL 字段添加到 Argo CD 操作对象的 **.status** 字段中。当为路由启用了路由或入口时，新的 URL 字段会显示路由。如果没有从路由或入口提供 URL，则不会显示 **.host** 字段。
  当配置了路由或入口时，但对应的控制器没有正确设置，且不是 **Ready** 状态，也不会传播其 URL，操作对象中的 **.status.host** 字段的值表示为 **Pending** 而不是显示 URL。这会影响操作对象的整体状态，方法是将其设置为 **Pending** 而不是 **Available**。GITOPS-654

5.1.26.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，特定于 **AppProjects** 的 RBAC 规则不允许在角色的 subject 字段中使用逗号，从而防止绑定到 LDAP 帐户。在这个版本中解决了这个问题，您可以在 **AppProject** 特定的 RBAC 规则中指定复杂的角色绑定。GITOPS-1771
- 在此次更新之前，当将 **DeploymentConfig** 资源扩展到 **0** 时，Argo CD 会显示它为 **progressing** 状态，并具有一个 **"replication controller is waiting for pods to run"**的健康状态信息。在这个版本中解决了边缘情况，健康检查现在会报告 **DeploymentConfig** 资源的正确健康状况。GITOPS-1738
- 在此次更新之前，Red Hat OpenShift GitOps 会删除 **argocd-tls-certs-cm** 配置映射中的 TLS 证书，除非证书是在 **ArgoCD** CR specification **tls.initialCerts** 字段中配置的。这个问题现已解决。GITOPS-1725
- 在此次更新之前，当使用 **managed-by** 标签创建命名空间时，它会在新命名空间中创建了大量 **RoleBinding** 资源。在这个版本中解决了这个问题，Red Hat OpenShift GitOps 会删除之前版本创建的不相关的 **Role** 和 **RoleBinding** 资源。GITOPS-1550
- 在此次更新之前，pass-through 模式中的路由的 TLS 证书没有 CA 名称。因此，Firefox 94 和更高版本无法连接到 Argo CD UI，并显示代码 **SEC_ERROR_BAD_DER**。在这个版本中解决了这个问题。您必须删除 **<openshift-gitops-ca>** secret 并使其重新创建。然后，您必须删除 **<openshift-gitops-tls>** secret。在 Red Hat OpenShift GitOps 重新创建后，Firefox 可以再次访问 Argo CD UI。GITOPS-1548

5.1.26.3. 已知问题

- 当使用 **Ingress** 资源而不是 OpenShift 集群中的 **Route** 资源时，Argo CD **.status.host** 字段不会被更新。GITOPS-1920

5.1.27. Red Hat OpenShift GitOps 1.4.13 发行注记

Red Hat OpenShift GitOps 1.4.13 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.27.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在 OpenShift Container Platform 4.12 中，安装控制台是可选的。在这个版本中，更新了 Red Hat OpenShift GitOps Operator，以防止安装控制台时出现 Operator 错误。GITOPS-2354

5.1.28. Red Hat OpenShift GitOps 1.4.12 发行注记

Red Hat OpenShift GitOps 1.4.12 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.28.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，因为存活度探测的无响应，在应用程序控制器的大型应用程序中会多次重启。在这个版本中，通过删除应用程序控制器 **StatefulSet** 对象中的存活度探测解决了这个问题。GITOPS-2153

- 在此次更新之前，当使用不受证书颁发机构签名的证书设置时，将无法验证 RHSSO 证书。在这个版本中解决了这个问题，您可以提供一个自定义证书，它会在与它通信时验证 Keycloak 的 TLS 证书。您可以将 **rootCA** 添加到 Argo CD 自定义资源 **.spec.keycloak.rootCA** 字段中。Operator 使用 PEM 编码的 root 证书协调此更改并更新 **argocd-cm ConfigMap** 中的 **oidc.config** 字段。GITOPS-2214

  注意

  REST在更新 **.spec.keycloak.rootCA** 字段后重启 Argo CD 服务器 pod。

  例如：

  ```
  apiVersion: argoproj.io/v1alpha1
  kind: ArgoCD
  metadata:
    name: example-argocd
    labels:
      example: basic
  spec:
    sso:
      provider: keycloak
      keycloak:
       rootCA: |
         ---- BEGIN CERTIFICATE ----
         This is a dummy certificate
         Please place this section with appropriate rootCA
         ---- END CERTIFICATE ----
    server:
      route:
        enabled: true
  ```

- 在此次更新之前，由 Argo CD 管理的终止命名空间会阻止创建角色和其他受管命名空间配置。在这个版本中解决了这个问题。GITOPS-2276

- 在此次更新之前，当将 **anyuid** 的 SCC 分配给 Dex **ServiceAccount** 资源时，Dex Pod 无法启动 **CreateContainerConfigError**。在这个版本中，通过将默认用户 ID 分配给 Dex 容器解决了这个问题。GITOPS-2235

5.1.29. Red Hat OpenShift GitOps 1.4.11 发行注记

Red Hat OpenShift GitOps 1.4.11 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.29.1. 新功能

当前发行版本包括以下改进：

- 在这个版本中，捆绑的 Argo CD 更新至 2.2.12 版本。

5.1.29.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当集群中存在更严格的 SCC 时，ArgoCD 实例的 **redis-ha-haproxy** Pod 会失败。在这个版本中，通过更新工作负载中的安全上下文解决了这个问题。GITOPS-2034

5.1.29.3. 已知问题

- Red Hat OpenShift GitOps Operator 可以使用带有 OIDC 和 Dex 的 RHSSO (KeyCloak)。但是，在应用了最新的安全修复程序时，Operator 无法验证 RHSSO 证书。GITOPS-2214
  作为临时解决方案，请在 ArgoCD 规格中禁用 OIDC (Keycloak/RHSSO) 端点的 TLS 验证。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
...
```

5.1.30. Red Hat OpenShift GitOps 1.4.6 发行注记

Red Hat OpenShift GitOps 1.4.6 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.30.1. 修复的问题

在当前发行版本中解决了以下问题：

- 基础镜像更新至最新版本，以避免 OpenSSL 缺陷链接：(CVE-2022-0778)。

注意

要安装 Red Hat OpenShift GitOps 1.4 的当前发行版本，并在其产品生命周期中接收进一步的更新，请切换到 **GitOps-1.4** 频道。

5.1.31. Red Hat OpenShift GitOps 1.4.5 发行注记

Red Hat OpenShift GitOps 1.4.5 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.31.1. 修复的问题

> **警告**
>
> 您应该直接从 Red Hat OpenShift GitOps v1.4.3 升级到 Red Hat OpenShift GitOps v1.4.5。在生产环境中不要使用 Red Hat OpenShift GitOps v1.4.4。影响 Red Hat OpenShift GitOps v1.4.4 的主要问题在 Red Hat OpenShift GitOps 1.4.5 中解决。

在当前发行版本中解决了以下问题：

- 在此次更新之前，Argo CD pod 处于 **ErrImagePullBackOff** 状态。显示以下出错信息：

```
reason: ErrImagePull
        message: >-
          rpc error: code = Unknown desc = reading manifest
          sha256:ff4ad30752cf0d321cd6c2c6fd4490b716607ea2960558347440f2f370a586a8
          in registry.redhat.io/openshift-gitops-1/argocd-rhel8: StatusCode:
          404, <HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>
```

这个问题现已解决。GITOPS-1848

5.1.32. Red Hat OpenShift GitOps 1.4.3 的发行注记

Red Hat OpenShift GitOps 1.4.3 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.32.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，Red Hat OpenShift GitOps 会删除 **argocd-tls-certs-cm** 配置映射中的 TLS 证书，除非证书是在 ArgoCD CR specification **tls.initialCerts** 字段中配置的。在这个版本中解决了这个问题。GITOPS-1725

5.1.33. Release notes for Red Hat OpenShift GitOps 1.4.2

Red Hat OpenShift GitOps 1.4.2 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.33.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，如果将多个 **Ingress** 附加到路由，则 **Route** 资源会处于 **Progressing** Health 状态。在这个版本中修复了健康检查，并报告 **Route** 资源的正确健康状况。GITOPS-1751

5.1.34. Red Hat OpenShift GitOps 1.4.1 发行注记

Red Hat OpenShift GitOps 1.4.1 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.34.1. 修复的问题

在当前发行版本中解决了以下问题：

- Red Hat OpenShift GitOps Operator v1.4.0 引入了一个回归问题，它从以下 CRD 的 **spec** 中删除描述字段：
  - **argoproj.io_applications.yaml**
  - **argoproj.io_appprojects.yaml**
  - **argoproj.io_argocds.yaml**
    在此次更新之前，当使用 **oc create** 命令创建 **AppProject** 资源时，因为缺少描述字段，资源无法同步。在这个版本中，恢复上述 CRD 中缺少的描述字段。GITOPS-1721

5.1.35. Red Hat OpenShift GitOps 1.4.0 发行注记

Red Hat OpenShift GitOps 1.4.0 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.10 中。

5.1.35.1. 新功能

当前版本添加了以下改进。

- 此功能增强将 Red Hat OpenShift GitOps Application Manager CLI (**kam**) 升级到 **0.0.41** 版本。GITOPS-1669
- 此增强将 Argo CD 升级到 **2.2.2** 版本。GITOPS-1532
- 此增强将 Helm 升级到 **3.7.1** 版本。GITOPS-1530
- 此功能增强将 **DeploymentConfig**、**Route** 和 **OLM Operator** 项目的健康状态添加到 Argo CD Dashboard 和 OpenShift Container Platform web 控制台中。这些信息可帮助您监控应用程序的整体健康状况。GITOPS-655, GITOPS-915, GITOPS-916, GITOPS-1110
- 在这个版本中，您可以为 **argocd-server** 和 **argocd-repo-server** 组件指定需要的副本数量，方法是在 Argo CD 自定义资源中分别指定 **.spec.server.replicas** 和 **.spec.repo.replicas** 属性。如果为 **argocd-server** 组件配置 pod 横向自动扩展(HPA)，它将优先于 Argo CD 自定义资源属性。GITOPS-1245
- 以管理用户身份，当使用 **argocd.argoproj.io/managed-by** 标签为命名空间提供 Argo CD 访问权限时，它会假定 namespace-admin 权限。这些特权是管理员向非管理员用户提供命名空间（如开发团队）的问题，因为特权使非管理员用户能够修改网络策略等对象。
  在这个版本中，管理员可以为所有受管命名空间配置通用集群角色。在 Argo CD 应用程序控制器的角色绑定中，Operator 指的是 **CONTROLLER_CLUSTER_ROLE** 环境变量。在 Argo CD 服务器的角色绑定中，Operator 指的是 **SERVER_CLUSTER_ROLE** 环境变量。如果这些环境变量包含自定义角色，Operator 不会创建默认的 admin 角色。相反，它将现有自定义角色用于所有受管命名空间。GITOPS-1290
- 在这个版本中，OpenShift Container Platform **Developer** 视角中的 **Environments** 页面会显示一个有问题的核心图标，用于指示降级资源，但不包括其状态为 **Progressing**、**Missing** 和 **Unknown** 的资源。控制台会显示一个黄色符号图标，以指示没有同步的资源。GITOPS-1307

5.1.35.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，当访问到 Red Hat OpenShift GitOps Application Manager CLI (**kam**) 的路由时，无需向用户显示任何有用信息的默认页面。在这个版本中解决了这个问题，默认页面会显示 **kam** CLI 的下载链接。GITOPS-923
- 在此次更新之前，在 Argo CD 自定义资源的命名空间中设置资源配额可能会导致 Red Hat SSO(RH SSO)实例的设置失败。在这个版本中，通过为 RH SSO 部署 pod 设置最小资源请求来解决这个问题。GITOPS-1297
- 在此次更新之前，如果您更改了 **argocd-repo-server** 工作负载的日志级别，Operator 不会协调此设置。这个问题的临时解决方案是删除部署资源，以便 Operator 使用新的日志级别重新创建它。在这个版本中，日志级别被正确地协调为现存的 **argocd-repo-server** 工作负载。GITOPS-1387
- 在此次更新之前，如果 Operator 管理了一个在 **argocd-secret** Secret 中缺少 **.data** 字段的 Argo CD 实例，则该实例的 Operator 会崩溃。在这个版本中解决了这个问题，因此当缺少 **.data** 字段时 Operator 不会崩溃。相反，secret 会重新生成，并且重新部署 **gitops-operator-controller-manager** 资源。GITOPS-1402
- 在此次更新之前，**gitopsservice** 服务被标注为内部对象。在这个版本中，删除了注解，以便更新或删除默认 Argo CD 实例，并使用 UI 在基础架构节点上运行 GitOps 工作负载。GITOPS-1429

5.1.35.3. 已知问题

当前发行版本中已知的问题：

- 如果从 Dex 验证供应商迁移到 Keycloak 供应商，您可能会遇到 Keycloak 的登录问题。
  要防止这个问题，在迁移时，从 Argo CD 自定义资源中删除 **.spec.dex** 部分来卸载 Dex。等待几分钟，让 Dex 完全卸载。然后，通过将 **.spec.sso.provider: keycloak** 添加到 Argo CD 自定义资源来安装 Keycloak。

  作为临时解决方案，通过删除 **.spec.sso.provider: keycloak** 来卸载 Keycloak。然后重新安装它。GITOPS-1450,GITOPS-1331

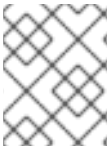
5.1.36. Red Hat OpenShift GitOps 1.3.7 发行注记

Red Hat OpenShift GitOps 1.3.7 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.6 中，且支持有限的 GA。

5.1.36.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在此次更新之前，在 OpenSSL 中发现了一个安全漏洞。在这个版本中解决了这个问题，将基础镜像更新至最新版本，以避免 OpenSSL 缺陷。(CVE-2022-0778).



注意

要安装 Red Hat OpenShift GitOps 1.3 的当前发行版本，并在其产品生命周期中接收进一步的更新，请切换到 **GitOps-1.3** 频道。

5.1.37. Red Hat OpenShift GitOps 1.3.6 发行注记

Red Hat OpenShift GitOps 1.3.6 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.6 中，且支持有限的 GA。

5.1.37.1. 修复的问题

在当前发行版本中解决了以下问题：

- 在 Red Hat OpenShift GitOps 中，不正确的访问控制允许 admin 权限升级 (CVE-2022-1025)。在这个版本中解决了这个问题。
- 路径遍历漏洞允许泄漏越界文件 (CVE-2022-24731)。在这个版本中解决了这个问题。
- 路径遍历缺陷以及不正确的访问控制允许泄漏越界文件 (CVE-2022-24730)。在这个版本中解决了这个问题。

5.1.38. Red Hat OpenShift GitOps 1.3.2 发行注记

Red Hat OpenShift GitOps 1.3.2 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.6 中，且支持有限的 GA。

5.1.38.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift GitOps 1.3.2 中的新内容。

- 将 Argo CD 升级到 **2.1.8**
- 将 Dex 升级到 **2.30.0**

5.1.38.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在以前的版本中，在 **Infrastructure Features** 部分的 OperatorHub UI 中，当您通过 **Disconnected** 过滤，Red Hat OpenShift GitOps Operator 不会显示在搜索结果中，因为 Operator 在 CSV 文件中没有设置相关的注解。在这个版本中，**Disconnected Cluster** 注解被添加到 Red Hat OpenShift GitOps Operator 中作为基础架构功能。GITOPS-1539
- 当使用 **Namespace-scoped** Argo CD 实例时，例如：一个没有限定到集群中的**所有命名空间**的 Argo CD 实例，Red Hat OpenShift GitOps 会动态维护一个受管命名空间列表。这些命名空间包括 **argocd.argoproj.io/managed-by** 标签。此命名空间列表存储在 **Argo CD → Settings → Clusters → "in-cluster" → NAMESPACES** 的缓存中。在此次更新之前，如果您删除了其中一个命名空间，Operator 会忽略该命名空间，命名空间会保留在列表中。这个行为会破坏集群配置中的 **CONNECTION STATE**，所有同步尝试都会导致错误。例如：

  > Argo service account does not have <random_verb> on <random_resource_type> in namespace <the_namespace_you_deleted>.

  这个程序错误已被解决。GITOPS-1521
- 在这个版本中，Red Hat OpenShift GitOps Operator 标注了 **Deep Insights** 功能级别。GITOPS-1519

- 在以前的版本中，Argo CD Operator 会自行管理 **resource.exclusion** 字段，但忽略 **resource.inclusion** 字段。这可以防止在 **Argo CD** CR 中配置的 **resource.inclusion** 项在 **argocd-cm** 配置映射中生成。这个程序错误已被解决。GITOPS-1518

5.1.39. Red Hat OpenShift GitOps 1.3.1 发行注记

Red Hat OpenShift GitOps 1.3.1 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.6 中，且支持有限的 GA。

5.1.39.1. 修复的问题

- 如果您升级到 v1.3.0，Operator 不会返回排序的环境变量片段。因此，协调器失败会导致在代理后运行的 OpenShift Container Platform 集群中频繁重新创建 Argo CD pod。在这个版本中解决了这个问题，使得 Argo CD pod 不会被重新创建。GITOPS-1489

5.1.40. Red Hat OpenShift GitOps 1.3 发行注记

Red Hat OpenShift GitOps 1.3 现在包括在 OpenShift Container Platform 4.7、4.8、4.9 和 4.6 中，且支持有限的 GA。

5.1.40.1. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift GitOps 1.3.0 中的新内容：

- 对于 v1.3.0 的全新安装，将自动配置 Dex。您可以使用 OpenShift 或 **kubeadmin** 凭证登录 **openshift-gitops** 命名空间中的默认 Argo CD 实例。作为 admin，您可以在安装 Operator 后禁用 Dex 安装，该 Operator 将从 **openshift-gitops** 命名空间中删除 Dex 部署。
- Operator 安装的默认 Argo CD 实例以及附带的控制器现在可以通过设置一个简单的配置切换在集群的基础架构节点上运行。
- Argo CD 中的内部通信现在可以使用 TLS 和 OpenShift 集群证书进行保护。Argo CD 路由现在除了使用外部证书管理器（如 cert-manager）外，还可以利用 OpenShift 集群证书。
- 使用控制台 4.9 的 **Developer** 视角中的 **Environments** 页面来深入了解 GitOps 环境。
- 现在，您可以使用 OLM 访问 Argo CD 中关于 **DeploymentConfig** 资源、**Route** 资源和 Operator 的自定义健康检查。
- GitOps Operator 现在符合最新 Operator-SDK 推荐的命名约定：
  - 前缀 **gitops-operator-** 添加到所有资源中
  - Service account 被重命名为 **gitops-operator-controller-manager**

5.1.40.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在以前的版本中，如果您将新命名空间设置为由 Argo CD 的新实例管理，它会立即变为**不同步** 状态，因为 Operator 创建用于管理该新命名空间的新角色和绑定。这个行为已被解决。GITOPS-1384

5.1.40.3. 已知问题

- 从 Dex 身份验证供应商迁移到 Keycloak 提供程序时，您可能会遇到 Keycloak 登录问题。GITOPS-1450
  为防止上述问题，在迁移时，通过删除 Argo CD 自定义资源中找到的 **.spec.dex** 部分来卸载 Dex。等待几分钟以便 Dex 完全卸载，然后通过将 **.spec.sso.provider: keycloak** 添加到 Argo CD 自定义资源来继续安装 Keycloak。

  作为临时解决方案，通过删除 **.spec.sso.provider: keycloak** 来卸载 Keycloak，然后重新安装。

5.1.41. Red Hat OpenShift GitOps 1.2.2 发行注记

Red Hat OpenShift GitOps 1.2.2 现在包括在 OpenShift Container Platform 4.8 中。

5.1.41.1. 修复的问题

在当前发行版本中解决了以下问题：

- 所有版本的 Argo CD 都容易受到一个路径遍历程序错误的影响，该程序错误地允许 Helm chart 使用任意值。在这个版本中解决了 CVE-2022-24348 gitops 错误，在传递 Helm 值文件时，路径遍历和解引用符号链接。GITOPS-1756

5.1.42. Red Hat OpenShift GitOps 1.2.1 发行注记

Red Hat OpenShift GitOps 1.2.1 现在包括在 OpenShift Container Platform 4.8 中。

5.1.42.1. 支持列表

这个版本中的一些功能当前还处于技术预览状态。它们并不适用于在生产环境中使用。

技术预览功能支持范围

在下表中，功能被标记为以下状态：

- **TP**: *技术预览*
- **GA**: *正式发行*

请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

表 5.2. 支持列表

功能	Red Hat OpenShift GitOps 1.2.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.42.2. 修复的问题

在当前发行版本中解决了以下问题：

- 在以前的版本中，应用程序控制器启动时会在应用程序控制器中观察到大量内存高峰。现在，应用程序控制器的 **--kubectl-parallelism-limit** 标志默认设置为 10，但可以通过在 Argo CD CR 规格中为 **.spec.controller.kubeParallelismLimit** 指定数字来覆盖这个值。GITOPS-1255

- 最新的 Triggers API 会导致在使用 **kam bootstrap** 命令时因为 kustomization.yaml 中的重复条目导致 Kubernetes 构建失败。Pipelines 和 Tekton 会在 v0.24.2 和 v0.14.2 中分别更新来解决这个问题。GITOPS-1273

- 现在，当从源命名空间中删除 Argo CD 实例时，持久化 RBAC 角色和绑定会自动从目标命名空间中移除。GITOPS-1228

- 在以前的版本中，当将 Argo CD 实例部署到命名空间中时，Argo CD 实例会将"managed-by"标签更改为自己的命名空间。在这个版本中，命名空间会取消标记，同时确保为命名空间创建和删除所需的 RBAC 角色和绑定。GITOPS-1247

- 在以前的版本中，Argo CD 工作负载中的默认资源请求限制（特别是 repo-server 和应用程序控制器）被发现非常严格的限制。现有资源配额现已被删除，在仓库服务器中默认内存限值已增加到 1024M。请注意，这个更改只会影响新的安装；现有的 Argo CD 实例工作负载不会受到影响。GITOPS-1274

5.1.43. Red Hat OpenShift GitOps 1.2 发行注记

Red Hat OpenShift GitOps 1.2 现在包括在 OpenShift Container Platform 4.8 中。

5.1.43.1. 支持列表

这个版本中的一些功能当前还处于技术预览状态。它们并不适用于在生产环境中使用。

技术预览功能支持范围

在下表中，功能被标记为以下状态：

- **TP**: *技术预览*

- **GA**: *正式发行*

请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

表 5.3. 支持列表

功能	Red Hat OpenShift GitOps 1.2
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.43.2. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift GitOps 1.2 中的新内容。

- 如果您没有对 openshift-gitops 命名空间的读写访问权限，现在可以使用 GitOps Operator 中的 **DISABLE_DEFAULT_ARGOCD_INSTANCE** 环境变量，并将值设置为 **TRUE** 以防止默认的 Argo CD 实例从 **openshift-gitops** 命名空间启动。

- 现在，在 Argo CD 工作负载中配置了资源请求和限制。在 **openshift-gitops** 命名空间中启用资源配额。因此，手动在 openshift-gitops 命名空间中部署的带外工作负载必须使用资源请求和限制进行配置，并且可能需要增加资源配额。

- Argo CD 身份验证现已与红帽 SSO 集成，并在集群中自动配置 OpenShift 4 身份提供商。此功能默认为禁用。要启用红帽 SSO，请在 **ArgoCD** CR 中添加 SSO 配置，如下所示。目前，**keycloak** 是唯一受支持的提供程序。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
     enabled: true
```

- 现在，您可以使用路由标签定义主机名来支持路由器分片。现在，支持在 **server** (argocd server)、**grafana** 和 **prometheus** 路由上设置标签。要在路由上设置标签，请在 **ArgoCD** CR 中的服务器的路由配置下添加标签。

  在 argocd 服务器上设置标签的 ArgoCD CR YAML 示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  server:
    route:
     enabled: true
     labels:
       key1: value1
       key2: value2
```

- GitOps Operator 现在会自动向 Argo CD 实例授予权限，以通过应用标签来管理目标命名空间中的资源。用户可以使用标签 **argocd.argoproj.io/managed-by: <source-namespace>** 标记目标命名空间，其中 **source-namespace** 是部署 argocd 实例的命名空间。

5.1.43.3. 修复的问题

在当前发行版本中解决了以下问题：

- 在以前的版本中，如果用户在 openshift-gitops 命名空间中创建了由默认集群实例管理的 Argo CD 实例，则负责新 Argo CD 实例的应用程序会停留在 **OutOfSync** 状态。现在，通过添加对集群 secret 的所有者引用解决了这个问题。GITOPS-1025

5.1.43.4. 已知问题

Red Hat OpenShift GitOps 1.2 中已知的问题。

- 当从源命名空间中删除 Argo CD 实例时，目标命名空间中的 **argocd.argoproj.io/managed-by** 标签不会被删除。GITOPS-1228
- Red Hat OpenShift GitOps 1.2 中的 openshift-gitops 命名空间中启用了资源配额。这会影响手动部署的带外工作负载，以及 **openshift-gitops** 命名空间中默认 Argo CD 实例部署的工作负载。当您从 Red Hat OpenShift GitOps **v1.1.2** 升级到 **v1.2** 时，此类工作负载必须使用资源请求和限制来配置。如果存在额外的工作负载，则必须增加 openshift-gitops 命名空间中的资源配额。**openshift-gitops** 命名空间的当前资源配额。

资源	Requests	Limits
CPU	6688m	13750m
内存	4544Mi	9070Mi

您可以使用以下命令来更新 CPU 限值。

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type='json' -p='[{"op": "replace", "path": "/spec/hard/limits.cpu", "value":"9000m"}]'
```

您可以使用以下命令来更新 CPU 请求。

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type='json' -p='[{"op": "replace", "path": "/spec/hard/cpu", "value":"7000m"}]
```

您可以替换以上命令中的路径从 **cpu** 到 **memory** 来更新内存。

5.1.44. Red Hat OpenShift GitOps 1.1 发行注记

Red Hat OpenShift GitOps 1.1 现在包括在 OpenShift Container Platform 4.7 中。

5.1.44.1. 支持列表

这个版本中的一些功能当前还处于技术预览状态。它们并不适用于在生产环境中使用。

技术预览功能支持范围

在下表中，功能被标记为以下状态：

- **TP**: *技术预览*
- **GA**: *正式发行*

请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

表 5.4. 支持列表

功能	Red Hat OpenShift GitOps 1.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.44.2. 新功能

除了包括修复和稳定性改进的信息外，以下小节突出介绍了 Red Hat OpenShift GitOps 1.1 中的新内容：

- 现在添加了 **ApplicationSet** 功能（技术预览）。**ApplicationSet** 功能可在大量集群和 monorepos 中管理 Argo CD 应用程序时实现自动化和更大的灵活性。它还可在多租户 Kubernetes 集群中实现自助服务。
- Argo CD 现在与集群日志记录堆栈以及 OpenShift Container Platform Monitoring 和 Alerting 功能集成。
- Argo CD auth 现在与 OpenShift Container Platform 集成。
- Argo CD 应用程序控制器现在支持横向扩展。
- Argo CD Redis 服务器现在支持高可用性（HA）。

5.1.44.3. 修复的问题

在当前发行版本中解决了以下问题：

- 在以前的版本中，Red Hat OpenShift GitOps 在带有活跃全局代理设置的代理服务器设置中无法正常工作。这个问题已被解决。现在，Red Hat OpenShift GitOps Operator 会使用 pod 的完全限定域名（FQDN）配置 Argo CD，以启用组件间的通信。GITOPS-703
- Red Hat OpenShift GitOps 后端依赖于 Red Hat OpenShift GitOps URL 中的 **?ref=** 查询参数来发出 API 调用。在以前的版本中，这个参数没有从 URL 中读取，从而导致后端始终考虑默认引用。这个问题已被解决。Red Hat OpenShift GitOps 后端现在从 Red Hat OpenShift GitOps URL 提取引用查询参数，且仅在未提供输入引用时使用默认引用。GITOPS-817
- 在以前的版本中，Red Hat OpenShift GitOps 后端无法找到有效的 GitLab 存储库。这是因为 Red Hat OpenShift GitOps 后端检查 **main** 作为分支引用，而不是 GitLab 存储库中的 **master**。这个问题现已解决。GITOPS-768
- OpenShift Container Platform Web 控制台的 **Developer** 视角中的 **Environments** 页面现在显示应用程序列表和环境数量。本页还显示 Argo CD 链接，它将您定向到列出所有应用程序的 Argo CD **Applications** 页面。Argo CD **Applications** 页面带有 **LABELS**（如 **app.kubernetes.io/name=appName**），它只帮助您过滤您选择的应用程序。GITOPS-544

5.1.44.4. 已知问题

Red Hat OpenShift GitOps 1.1 中已知的问题：

- Red Hat OpenShift GitOps 不支持 Helm v2 和 ksonnet。
- 在断开连接的集群中不支持 Red Hat SSO（RH SSO）Operator。因此，断开连接的集群中不支持 Red Hat OpenShift GitOps Operator 和 RH SSO 集成。
- 当您从 OpenShift Container Platform web 控制台删除 Argo CD 应用程序时，Argo CD 应用程序会从用户界面中删除，但部署仍存在于集群中。作为临时解决方案，请从 Argo CD 控制台删除 Argo CD 应用程序。GITOPS-830

5.1.44.5. 有问题的更改

5.1.44.5.1. 从 Red Hat OpenShift GitOps v1.0.1 升级

当您从 Red Hat OpenShift GitOps **v1.0.1** 升级到 **v1.1** 时，Red Hat OpenShift GitOps Operator 会将 **openshift-gitops** 命名空间中创建的默认 Argo CD 实例从 **argocd-cluster** 重命名到 **openshift-gitops**。

这是一个有问题的变化，需要在升级前手动执行以下步骤：

1. 进入 OpenShift Container Platform web 控制台，将 **openshift-gitops** 命名空间中的 **argocd-cm.yml** 配置映射文件的内容复制到本地文件中。内容可能类似以下示例：

   argocd 配置映射 YAML 示例

```
kind: ConfigMap
apiVersion: v1
metadata:
selfLink: /api/v1/namespaces/openshift-gitops/configmaps/argocd-cm
resourceVersion: '112532'
name: argocd-cm
uid: f5226fbc-883d-47db-8b53-b5e363f007af
creationTimestamp: '2021-04-16T19:24:08Z'
managedFields:
...
namespace: openshift-gitops
labels:
  app.kubernetes.io/managed-by: argocd-cluster
  app.kubernetes.io/name: argocd-cm
  app.kubernetes.io/part-of: argocd
data: "" (1)
admin.enabled: 'true'
statusbadge.enabled: 'false'
resource.exclusions: |
  - apiGroups:
    - tekton.dev
    clusters:
    - '*'
    kinds:
    - TaskRun
    - PipelineRun
ga.trackingid: ''
repositories: |
  - type: git
    url: https://github.com/user-name/argocd-example-apps
ga.anonymizeusers: 'false'
help.chatUrl: ''
```

```
url: >-
  https://argocd-cluster-server-openshift-gitops.apps.dev-svc-4.7-
041614.devcluster.openshift.com   "" 2
help.chatText: ''
kustomize.buildOptions: ''
resource.inclusions: ''
repository.credentials: ''
users.anonymous.enabled: 'false'
configManagementPlugins: ''
application.instanceLabelKey: ''
```

1 手工恢复 **argocd-cm.yml** 配置映射文件中内容的 **data** 部分。

2 将配置映射条目中的 URL 值替换为新实例名称 **openshift-gitops**。

2. 删除默认的 **argocd-cluster** 实例。
3. 编辑新的 **argocd-cm.yml** 配置映射文件，手动恢复整个 **data** 部分。
4. 将配置映射条目中的 URL 值替换为新实例名称 **openshift-gitops**。例如，在上例中将 URL 值替换为以下 URL 值：

```
url: >-
  https://openshift-gitops-server-openshift-gitops.apps.dev-svc-4.7-
041614.devcluster.openshift.com
```

5. 登录到 Argo CD 集群，验证之前的配置是否存在。

5.2. 了解 OPENSHIFT GITOPS

5.2.1. 关于 GitOps

GitOps 是为云原生应用程序实施持续部署的一种声明方式。您可以使用 GitOps 创建可重复进程，用于在多集群 Kubernetes 环境间管理 OpenShift Container Platform 集群和应用程序。GitOps 以快速的速度处理和自动化复杂部署，节省部署和发行周期期间的时间。

GitOps 工作流通过开发、测试、临时和生产环境来推送应用程序。GitOps 部署新应用程序或更新现有应用程序，因此您只需要更新存储库，GitOps 会自动执行所有操作。

GitOps 是一组使用 Git 拉取请求来管理基础架构和应用程序配置的实践。GitOps 中的 Git 存储库是系统和应用程序配置的唯一来源。此 Git 存储库包含指定环境中所需的基础架构声明描述，并包含自动流程，以使您的环境与上述状态匹配。它还包含该系统的完整状态，以便可查看并可审核更改到系统状态。通过使用 GitOps，您可以处理基础架构和应用程序配置 sprawl 的问题。

GitOps 将基础架构和应用程序定义定义为代码。然后，它会使用此代码来管理多个工作区和集群来简化基础架构和应用程序配置的创建过程。根据代码原则，您可以在 Git 存储库中存储集群和应用程序的配置，然后按照 Git 工作流将这些存储库应用到所选集群中。您可以将在 Git 存储库中开发和维护软件的核心原则应用到创建和管理集群和应用程序配置文件。

5.2.2. 关于 Red Hat OpenShift GitOps

当应用程序部署到不同环境中的不同集群时，Red Hat OpenShift GitOps 可确保应用程序的一致性，如开发、临时和生产环境。Red Hat OpenShift GitOps 整理与配置仓库相关的部署过程，并将其作为核心元素。它总会保持至少有两个软件仓库：

1. 源代码的应用程序仓库
2. 定义应用程序所需状态的环境配置仓库

这些软件仓库包含您指定环境中所需的基础架构声明信息。它们还包含可让您的环境与上述状态匹配的自动过程。

Red Hat OpenShift GitOps 使用 Argo CD 来维护集群资源。Argo CD 是一个开源声明工具，用于应用程序的持续集成和持续部署（CI/CD）。Red Hat OpenShift GitOps 将 Argo CD 实现作为一个控制器，以便持续监控 Git 存储库中定义的应用程序定义和配置。然后，Argo CD 将这些配置的指定状态与集群中的实时状态进行比较。

Argo CD 报告与指定状态不同的配置。报告允许管理员自动或者手动将配置重新同步到定义的状态。因此，ArgoCD 可让您提供全局自定义资源，如用于配置 OpenShift Container Platform 集群的资源。

5.2.2.1. 主要特性

Red Hat OpenShift GitOps 可帮助您自动执行以下任务：

- 确保集群具有类似的配置、监控和存储状态
- 对多个 OpenShift Container Platform 集群应用或恢复配置更改
- 将模板配置与不同环境关联
- 在集群间（从调试到生产阶段）推广应用程序。

5.3. 安装 RED HAT OPENSHIFT GITOPS

Red Hat OpenShift GitOps 使用 Argo CD 管理特定集群范围的资源，包括：集群 Operator、可选 Operator Lifecycle Manager（OLM）Operator 和用户管理。

本指南介绍如何将 Red Hat OpenShift GitOps Operator 安装到 OpenShift Container Platform 集群，并登录 Argo CD 实例。

重要

latest 频道启用 Red Hat OpenShift GitOps Operator 最新稳定版本的安装。目前，这是安装 Red Hat OpenShift GitOps Operator 的默认频道。

要安装 Red Hat OpenShift GitOps Operator 的特定版本，集群管理员可以使用对应的 **gitops-<version>** 频道。例如，要安装 Red Hat OpenShift GitOps Operator 版本 1.8.x，您可以使用 **gitops-1.8** 频道。

5.3.1. 在 Web 控制台中安装 Red Hat OpenShift GitOps Operator

先决条件

- 访问 OpenShift Container Platform Web 控制台。
- 具有 **cluster-admin** 角色的帐户。

- 以管理员身份登陆到 OpenShift Container Platform 集群。

警告

如果您已安装 Argo CD Operator 的 Community 版本，请在安装 Red Hat OpenShift GitOps Operator 前删除 Argo CD Community Operator。

流程

1. 打开 Web 控制台的 **Administrator** 视角，并进入左侧菜单中的 **Operators → OperatorHub**。
2. 搜索 **OpenShift GitOps**，点 **Red Hat OpenShift GitOps** 标题，然后点 **Install**。
   Red Hat OpenShift GitOps 将安装在集群的所有命名空间中。

安装 Red Hat OpenShift GitOps Operator 后，它会自动设置 **openshift-gitops** 命名空间中的已就绪的 Argo CD 实例，并在控制台工具栏中显示 Argo CD 图标。您可以在项目下为您的应用程序创建后续的 Argo CD 实例。

5.3.2. 使用 CLI 安装 Red Hat OpenShift GitOps Operator

您可以使用 CLI 从 OperatorHub 安装 Red Hat OpenShift GitOps Operator。

流程

1. 创建一个 Subscription 对象 YAML 文件，以便为 Red Hat OpenShift GitOps 订阅一个命名空间，如 **sub.yaml** ：

   订阅示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
spec:
  channel: latest (1)
  installPlanApproval: Automatic
  name: openshift-gitops-operator (2)
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace (4)
```

(1) 指定您要订阅 Operator 的频道名称。

(2) 指定要订阅的 Operator 的名称。

(3) 指定提供 Operator 的 CatalogSource 的名称。

(4) CatalogSource 的命名空间。将 **openshift-marketplace** 用于默认的 OperatorHub CatalogSource。

2. 将订阅应用到集群：

```
$ oc apply -f openshift-gitops-sub.yaml
```

3. 安装完成后，确保 **openshift-gitops** 命名空间中的所有 pod 都在运行：

```
$ oc get pods -n openshift-gitops
```

输出示例

```
NAME                                                READY   STATUS RESTARTS   AGE
cluster-b5798d6f9-zr576                             1/1  Running   0       65m
kam-69866d7c48-8nsjv                                1/1  Running   0       65m
openshift-gitops-application-controller-0           1/1  Running   0       53m
openshift-gitops-applicationset-controller-6447b8dfdd-5ckgh 1/1  Running   0       65m
openshift-gitops-redis-74bd8d7d96-49bjf             1/1  Running   0       65m
openshift-gitops-repo-server-c999f75d5-l4rsg        1/1  Running   0       65m
openshift-gitops-server-5785f7668b-wj57t            1/1  Running   0       53m
```

5.3.3. 使用 Argo CD admin 帐户登录到 Argo CD 实例

Red Hat OpenShift GitOps Operator 会自动创建一个可用的 Argo CD 实例，可在 **openshift-gitops** 命名空间中使用。

先决条件

- 在集群中安装了 Red Hat OpenShift GitOps Operator。

流程

1. 在 Web 控制台的 **Administrator** 视角中，导航到 **Operators → Installed Operators**，以验证是否安装了 Red Hat OpenShift GitOps Operator。

2. 进入  menu → **OpenShift GitOps → Cluster Argo CD**。Argo CD UI 的登录页面显示在新窗口中。

3. 可选：要使用 OpenShift Container Platform 凭证登录，请确保您是 **cluster-admins** 组的用户，然后在 Argo CD 用户界面中选择 **LOG IN VIA OPENSHIFT** 选项。

注意

要是 **cluster-admins** 组的用户，请使用 **oc adm groups new cluster-admins <user&** gt; 命令，其中 < **user** > 是您可以绑定到集群范围或本地的用户和组的默认集群角色。

4. 要使用您的用户名和密码登录，请获取 Argo CD 实例的密码：

   a. 在控制台左侧面板中，使用视角切换器切换到 **Developer** 视角。

   b. 使用 **Project** 下拉列表，再选择 **openshift-gitops** 项目。

   c. 使用左侧导航面板导航到 **Secrets** 页面。

d. 选择 **openshift-gitops-cluster** 实例来显示密码。

e. 复制密码。

5. 使用此密码和 **admin** 作为用户名在新窗口中登录到 Argo CD UI。

注意

您不能在同一命名空间中创建两个 Argo CD CR。

5.4. 卸载 OPENSHIFT GITOPS

卸载 Red Hat OpenShift GitOps Operator 分为两个步骤：

1. 删除在 Red Hat OpenShift GitOps Operator 的默认命名空间中添加的 Argo CD 实例。
2. 卸载 Red Hat OpenShift GitOps Operator。

仅卸载 Operator 不会删除创建的 Argo CD 实例。

5.4.1. 删除 Argo CD 实例

删除添加到 GitOps Operator 命名空间中的 Argo CD 实例。

流程

1. 在 **Terminal** 中键入以下命令：

```
$ oc delete gitopsservice cluster -n openshift-gitops
```

注意

您无法从 Web 控制台 UI 删除 Argo CD 集群。

命令成功运行后，所有 Argo CD 实例都将从 **openshift-gitops** 命名空间中删除。

使用以下命令从其他命名空间删除任何其他 Argo CD 实例：

```
$ oc delete gitopsservice cluster -n <namespace>
```

5.4.2. 卸载 GitOps Operator

流程

1. 在 **Operators → OperatorHub** 页面中，使用 **Filter by keyword** 复选框来搜索 **Red Hat OpenShift GitOps Operator** 标题。
2. 点 **Red Hat OpenShift GitOps Operator** 标题。Operator 标题表示已安装该 Operator。
3. 在 **Red Hat OpenShift GitOps Operator** 描述符页面中，点 **Uninstall**。

其他资源

- 您可以参阅从集群中卸载 Operators一节中的内容来了解更多有关从 OpenShift Container Platform 上卸载 Operator 的信息。

5.5. 设置 ARGO CD 实例

默认情况下，Red Hat OpenShift GitOps 在 **openshift-gitops** 命名空间中安装 Argo CD 实例，并使用额外的权限来管理某些集群范围的资源。要管理集群配置或部署应用程序，您可以安装和部署新的 Argo CD 实例。默认情况下，任何新实例都只能管理部署它的命名空间中的资源。

5.5.1. 安装 Argo CD

要管理集群配置或部署应用程序，您可以安装和部署新的 Argo CD 实例。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 点 **Operators → Installed Operators**。
3. 从 **Project** 下拉菜单中选择您要安装 Argo CD 实例的项目。
4. 从已安装的 Operator 选择 **OpenShift GitOps Operator**，然后选择 **Argo CD** 选项卡。
5. 点 **Create** 配置参数：
   a. 输入实例的 **Name**。默认情况下，**Name** 被设置为 **argocd**。
   b. 创建外部操作系统路由来访问 Argo CD 服务器。点 **Server → Route** 并检查 **Enabled**。
6. 要打开 Argo CD web UI，请进入安装 Argo CD 实例的项目中的 **Networking → Routes → <instance name>-server**，点路由。

5.5.2. 为 Argo CD 服务器和存储库服务器启用副本

Argo CD-server 和 Argo CD-repo-server 工作负载是无状态的。要在 pod 中更好地分布工作负载，您可以增加 Argo CD-server 和 Argo CD-repo-server 副本的数量。但是，如果 Argo CD-server 上启用了 pod 横向自动扩展，它会覆盖您设置的副本数。

流程

- 将 **repo** 和 **server** spec 的 **replicas** 参数设置为您要运行的副本数：

  Argo CD 自定义资源示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  repo:
    replicas: <number_of_replicas>
  server:
    replicas: <number_of_replicas>
```

```
  route:
    enabled: true
    path: /
    tls:
      insecureEdgeTerminationPolicy: Redirect
      termination: passthrough
    wildcardPolicy: None
```

5.5.3. 将资源部署到不同的命名空间中

要允许 Argo CD 管理除安装它外的其他命名空间中的资源，请使用 **argocd.argoproj.io/managed-by** 标签配置目标命名空间。

流程

- 配置命名空间：

```
$ oc label namespace <namespace> \
argocd.argoproj.io/managed-by=<instance_name> 1
```

1 安装 Argo CD 的命名空间。

5.5.4. 自定义 Argo CD 控制台链接

在多租户集群中，用户可能需要处理多个 Argo CD 实例。例如，在命名空间中安装 Argo CD 实例后，您可能会在 Console Application Launcher 中找到附加到 Argo CD 控制台链接的不同 Argo CD 实例，而不是在 Console Application Launcher 中找到自己的 Argo CD 实例。

您可以通过设置 **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** 环境变量来自定义 Argo CD 控制台链接：

- 当您将 **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** 设置为 **true** 时，Argo CD 控制台链接将永久删除。
- 当您将 **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** 设置为 **false** 或使用默认值时，Argo CD 控制台链接会被临时删除并在 Argo CD 路由被协调时再次可见。

先决条件

- 以管理员身份登录到 OpenShift Container Platform 集群。
- 已安装 Red Hat OpenShift GitOps Operator。

流程

1. 在 **Administrator** 视角中，进入到 **Administration → CustomResourceDefinitions**。
2. 找到 **Subscription** CRD 并点它打开它。
3. 选择 **Instances** 选项卡，然后点 **openshift-gitops-operator** 订阅。
4. 选择 **YAML** 选项卡并进行自定义：

- 要启用或禁用 Argo CD 控制台链接，请根据需要编辑 **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** 的值：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
spec:
  config:
    env:
    - name: DISABLE_DEFAULT_ARGOCD_CONSOLELINK
      value: 'true'
```

5.6. 监控 ARGO CD 实例

默认情况下，Red Hat OpenShift GitOps Operator 会自动检测您定义的命名空间中已安装的 Argo CD 实例，如 **openshift-gitops**，并将其连接到集群的监控堆栈，以便为不同步应用程序提供警报。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。
- 访问 OpenShift Container Platform web 控制台。
- 在集群中安装了 Red Hat OpenShift GitOps Operator。
- 您已在定义的命名空间中安装了 Argo CD 应用程序，如 **openshift-gitops**。

5.6.1. 使用 Prometheus 指标监控 Argo CD 健康状况

您可以通过针对它运行 Prometheus 指标查询来监控 Argo CD 应用程序的健康状况。

流程

1. 在 web 控制台的 **Developer** 视角中，选择安装 Argo CD 应用程序的命名空间，并导航到 **Observe → Metrics**。
2. 在 **Select query** 下拉列表中，选择 **Custom query**。
3. 要检查 Argo CD 应用程序的健康状态，在 **Expression** 字段中输入 Prometheus Query Language (PromQL)查询，如下例所示：

   示例

```
sum(argocd_app_info{dest_namespace=~"<your_defined_namespace>",health_status!=""}) by (health_status) 1
```

1 将 &**lt;your_defined_namespace** > 变量替换为您定义的命名空间的实际名称，如 **openshift-gitops**。

5.7. 通过部署带有集群配置的应用程序来配置 OPENSHIFT 集群

使用 Red Hat OpenShift GitOps，您可以将 Argo CD 配置为将 Git 目录的内容与包含集群自定义配置的应用程序递归同步。

先决条件

- 以管理员身份登录到 OpenShift Container Platform 集群。
- 在集群中安装了 Red Hat OpenShift GitOps Operator。
- 已登录到 Argo CD 实例。

5.7.1. 使用 Argo CD 实例管理集群范围的资源

要管理集群范围的资源，请更新 Red Hat OpenShift GitOps Operator 的现有 **Subscription** 对象，并将 Argo CD 实例的命名空间添加到 **spec** 部分中的 **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 环境变量中。

流程

1. 在 Web 控制台的 **Administrator** 视角中，进入到 **Operators → Installed Operators → Red Hat OpenShift GitOps → Subscription**。
2. 点 **Actions** 下拉菜单，然后点 **Edit Subscription**。
3. 在 **openshift-gitops-operator** 订阅详情页面的 **YAML** 选项卡下，通过将 Argo CD 实例的命名空间添加到 **spec** 部分中的 **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 环境变量来编辑 **Subscription** YAML 文件：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
...
spec:
  config:
    env:
    - name: ARGOCD_CLUSTER_CONFIG_NAMESPACES
      value: openshift-gitops, <list of namespaces of cluster-scoped Argo CD instances>
...
```

4. 要验证 Argo 实例是否已配置有集群角色来管理集群范围的资源，请执行以下步骤：
   a. 进入到 **User Management → Roles**，然后从 **Filter** 下拉菜单中选择 **Cluster-wide Roles**。
   b. 使用 **Search by name** 字段搜索 **argocd-application-controller**。
   Roles 页面显示创建的集群角色。

提示

或者，在 OpenShift CLI 中运行以下命令：

```
oc auth can-i create oauth -n openshift-gitops --as system:serviceaccount:openshift-gitops:openshift-gitops-argocd-application-controller
```

输出 **yes** 代表 Argo 实例配置了集群角色来管理集群范围的资源。否则，检查您的配置并根据需要执行必要的步骤。

5.7.2. Argocd 实例的默认权限

默认情况下，Argo CD 实例具有以下权限：

- Argo CD 实例具有 **admin** 权限，以便仅管理部署它的命名空间中的资源。例如，在 **foo** 命名空间中部署的 Argo CD 实例具有 **admin** 权限，仅管理该命名空间的资源。
- Argo CD 具有以下集群范围的权限，因为 Argo CD 需要对资源进行集群范围的**读取**权限才能正常工作：

```
- verbs:
  - get
  - list
  - watch
 apiGroups:
  - '*'
 resources:
  - '*'
- verbs:
  - get
  - list
 nonResourceURLs:
  - '*'
```

注意

- 您可以编辑运行 Argo CD 的 **argocd-server** 和 **argocd-application-controller** 组件使用的集群角色，以便**写入**权限仅限于您希望 Argo CD 管理的命名空间和资源。

```
$ oc edit clusterrole argocd-server
$ oc edit clusterrole argocd-application-controller
```

5.7.3. 在集群级别运行 Argo CD 实例

默认 Argo CD 实例和附带的控制器（由 Red Hat OpenShift GitOps Operator 安装）现在可以通过设置一个简单的配置切换在集群的基础架构节点上运行。

流程

1. 标记现有节点：

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

2. 可选：如果需要，您还可以在基础架构节点上应用污点并隔离工作负载，并防止其他工作负载在这些节点上调度：

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra \
infra=reserved:NoSchedule infra=reserved:NoExecute
```

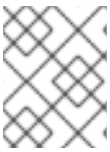
3. 在 **GitOpsService** 自定义资源中添加 **runOnInfra** 切换：

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. 可选：如果将污点添加到节点，则在 **GitOpsService** 自定义资源中添加 **容限**，例如：

```
spec:
  runOnInfra: true
  tolerations:
  - effect: NoSchedule
    key: infra
    value: reserved
  - effect: NoExecute
    key: infra
    value: reserved
```

5. 通过在控制台 UI 中查看 **Pods → Pod details**，验证 **openshift-gitops** 命名空间中的工作负载现在已调度到基础架构节点上。



注意

任何手工添加到默认 Argo CD 自定义资源中的 **nodeSelectors** 和 **tolerations**，都会被 **GitOpsService** 自定义资源的 **tolerations** 覆盖。

其他资源

- 如需有关污点和容限的更多信息，请参阅使用节点污点控制 pod 放置。
- 有关基础架构机器集的更多信息，请参阅创建基础架构机器集。

5.7.4. 使用 Argo CD 仪表板创建应用程序

Argo CD 提供了一个仪表板，供您创建应用程序。

此示例工作流逐步指导您完成将 Argo CD 配置为递归将 **cluster** 目录中的内容同步到 **cluster-configs-** 应用程序。目录定义了 OpenShift Container Platform Web 控制台集群配置，在 web 控制台中的  菜单下向 **Red Hat Developer Blog - Kubernetes**添加链接，并在集群中定义命名空间 **spring-petclinic**。

流程

1. 在 Argo CD 控制面板中，单击 **NEW APP** 以添加新 Argo CD 应用。

2. 对于此工作流，使用以下配置创建一个 **cluster-configs** 应用程序：

   应用程序名称
   cluster-configs
   project
   default
   同步策略
   Manual（手动）
   仓库 URL
   https://github.com/redhat-developer/openshift-gitops-getting-started
   修订
   HEAD
   路径
   cluster
   目的地
   https://kubernetes.default.svc
   命名空间
   spring-petclinic
   Directory Recurse
   checked

3. 单击 **CREATE** 以创建应用程序。

4. 打开 Web 控制台的 **Administrator** 视角，并导航到左侧菜单中的 **Administration → Namespaces**。

5. 搜索并选择命名空间，然后在 **Label** 字段中输入 **argocd.argoproj.io/managed-by=openshift-gitops**，以便 **openshift-gitops** 命名空间中的 Argo CD 实例可以管理您的命名空间。

5.7.5. 使用 oc 工具创建应用程序

您可以使用 **oc** 工具在终端中创建 Argo CD 应用程序。

流程

1. 下载 示例应用程序：

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. 创建应用程序：

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. 运行 **oc get** 命令以查看所创建的应用程序：

```
$ oc get application -n openshift-gitops
```

4. 在部署应用程序的命名空间中添加标签，以便 **openshift-gitops** 命名空间中的 Argo CD 实例可以管理它：

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

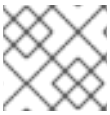
5.7.6. 将应用程序与 Git 存储库同步

流程

1. 在 Argo CD 仪表板中，**cluster-configs** Argo CD 应用程序的状态为 **Missing** 和 **OutOfSync**。因为应用程序配置了手动同步策略，所以 Argo CD 不会自动同步。
2. 点 **cluster-configs** 标题上的 **SYNC**，查看更改，然后点 **SYNCHRONIZE**。Argo CD 将自动检测 Git 存储库中的任何更改。如果更改了配置，Argo CD 会将 **cluster-configs** 的状态改为 **OutOfSync**。您可以修改 Argo CD 的同步策略，以自动将 Git 存储库中的更改应用到集群。
3. 现在，**cluster-configs** Argo CD 应用程序的状态为 **Healthy** 和 **Synced**。点 **cluster-configs** 标题检查同步资源的详情及其在集群中的状态。
4. 进入到 OpenShift Container Platform Web 控制台并点击  以验证 **Red Hat Developer Blog - Kubernetes** 的链接现在是否存在。
5. 导航到 **Project** 页面并搜索 **spring-petclinic** 命名空间，以验证它是否已添加到集群中。集群配置已成功与集群同步。

5.7.7. 集群配置的内置权限

默认情况下，Argo CD 实例具有管理特定集群范围资源的权限，如集群 Operator、可选 OLM Operator 和用户管理。



注意

Argo CD 没有 cluster-admin 权限。

Argo CD 实例的权限：

Resources	描述
资源组	配置用户或管理员
operators.coreos.com	由 OLM 管理的可选 Operator
user.openshift.io , **rbac.authorization.k8s.io**	组、用户及其权限
config.openshift.io	由 CVO 管理的 control plane Operator，用于配置集群范围的构建配置、registry 配置和调度程序策略
storage.k8s.io	存储

console.openshift.io	控制台自定义

5.7.8. 为集群配置添加权限

您可以授予 Argo CD 实例的权限来管理集群配置。创建具有额外权限的集群角色，然后创建新的集群角色绑定以将集群角色与服务帐户关联。

流程

1. 以 admin 用户身份登录 OpenShift Container Platform Web 控制台。
2. 在 Web 控制台中，选择 **User Management → Roles → Create Role**。使用以下 **ClusterRole** YAML 模板来添加规则来指定额外权限。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secrets-cluster-role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["*"]
```

3. 点 **Create** 添加集群角色。
4. 现在，创建集群角色绑定。在 Web 控制台中，选择 **User Management → Role Bindings → Create Binding**。
5. 从 **Project** 下拉菜单中选择 **All Projects**。
6. 点 **Create binding**。
7. 将 **Binding type** 选择为 **Cluster-wide role binding(ClusterRoleBinding)**。
8. 为 **RoleBinding 名称** 输入一个唯一值。
9. 从下拉列表中选择新创建的集群角色或现有集群角色。
10. 选择 **Subject** 作为 **ServiceAccount**，并提供 **Subject 命名空间**和**名称**。
    a. **主题命名空间**:**openshift-gitops**
    b. **主题名称**:**openshift-gitops-argocd-application-controller**
11. 点 **Create**。**ClusterRoleBinding** 对象的 YAML 文件如下：

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-role-binding
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
```

```
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: admin
```

5.7.9. 使用 Red Hat OpenShift GitOps 安装 OLM Operator

带有集群配置的 Red Hat OpenShift GitOps 管理特定集群范围的资源，并负责安装集群 Operator 或任何命名空间范围的 OLM Operator。

考虑作为集群管理员的情况，您必须安装 OLM Operator，如 Tekton。您可以使用 OpenShift Container Platform Web 控制台手动安装 Tekton Operator 或 OpenShift CLI，在集群中手动安装 Tekton 订阅和 Tekton Operator 组。

Red Hat OpenShift GitOps 将 Kubernetes 资源放置在 Git 存储库中。作为集群管理员，使用 Red Hat OpenShift GitOps 管理和自动化其他 OLM Operator 安装，而无需手动步骤。例如，在使用 Red Hat OpenShift GitOps 将 Tekton 订阅放在 Git 仓库后，Red Hat OpenShift GitOps 会自动从 Git 仓库获取此 Tekton 订阅，并在集群中安装 Tekton Operator。

5.7.9.1. 安装集群范围的 Operator

Operator Lifecycle Manager (OLM) 为集群范围的 Operator 使用 **openshift-operators** 命名空间中的默认 **global-operators** Operator 组。因此，您不必在 Gitops 仓库中管理 **OperatorGroup** 资源。但是，对于命名空间范围的 Operator，您必须管理该命名空间中的 **OperatorGroup** 资源。

要安装集群范围的 Operator，请在 Git 仓库中创建并放置所需 Operator 的 **Subscription** 资源。

示例：Grafana Operator 订阅

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: grafana
spec:
  channel: v4
  installPlanApproval: Automatic
  name: grafana-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

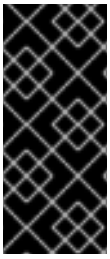
5.7.9.2. 安装 namepace-scoped Operator

要安装命名空间范围的 Operator，请在 Git 仓库中创建并放置所需 Operator 的 **Subscription** 和 **OperatorGroup** 资源。

示例：Ansible Automation Platform Resource Operator

```
...
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: ansible-automation-platform
...
```

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ansible-automation-platform-operator
  namespace: ansible-automation-platform
spec:
  targetNamespaces:
    - ansible-automation-platform
...
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ansible-automation-platform
  namespace: ansible-automation-platform
spec:
  channel: patch-me
  installPlanApproval: Automatic
  name: ansible-automation-platform-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
...
```



重要

当使用 Red Hat OpenShift GitOps 部署多个 Operator 时，您必须在对应的命名空间中创建单个 Operator 组。如果一个命名空间中存在多个 Operator 组，则在该命名空间中创建的任何 CSV 都会变为带有 **TooManyOperatorGroups** 原因的 **failure** 状态。在相应命名空间中的 Operator 组数量达到一个后，所有以前有 **failure** 状态的 CSV 都会过渡到 **pending** 状态。您需要手动批准待处理的安装计划以完成 Operator 安装。

5.8. 使用 ARGO CD 部署 SPRING BOOT 应用程序

通过 Argo CD，您可以使用 Argo CD 仪表板或使用 **oc** 工具将应用程序部署到 OpenShift 集群。

先决条件

- 在集群中安装了 Red Hat OpenShift GitOps。
- 登录到 Argo CD 实例。

5.8.1. 使用 Argo CD 仪表板创建应用程序

Argo CD 提供了一个仪表板，供您创建应用程序。

此示例工作流逐步指导您完成将 Argo CD 配置为递归将 **cluster** 目录中的内容同步到 **cluster-configs-** 应用程序。目录定义了 OpenShift Container Platform Web 控制台集群配置，在 web 控制台中的  菜单下向 **Red Hat Developer Blog - Kubernetes**添加链接，并在集群中定义命名空间 **spring-petclinic**。

流程

1. 在 Argo CD 控制面板中，单击 **NEW APP** 以添加新 Argo CD 应用。
2. 对于此工作流，使用以下配置创建一个 **cluster-configs** 应用程序：

应用程序名称

cluster-configs

project

default

同步策略

Manual（手动）

仓库 URL

https://github.com/redhat-developer/openshift-gitops-getting-started

修订

HEAD

路径

cluster

目的地

https://kubernetes.default.svc

命名空间

spring-petclinic

Directory Recurse

checked

3. 对于此工作流，使用以下配置创建一个 spring-petclinic 应用程序：

应用程序名称

spring-petclinic

project

default

同步策略

自动

仓库 URL

https://github.com/redhat-developer/openshift-gitops-getting-started

修订

HEAD

路径

app

目的地

https://kubernetes.default.svc

命名空间

spring-petclinic

4. 单击 CREATE 以创建应用程序。

5. 打开 Web 控制台的 Administrator 视角，并导航到左侧菜单中的 Administration → Namespaces。

6. 搜索并选择命名空间，然后在 Label 字段中输入 **argocd.argoproj.io/managed-by=openshift-gitops**，以便 **openshift-gitops** 命名空间中的 Argo CD 实例可以管理您的命名空间。

5.8.2. 使用 oc 工具创建应用程序

您可以使用 **oc** 工具在终端中创建 Argo CD 应用程序。

流程

1. 下载 示例应用程序：

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. 创建应用程序：

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. 运行 **oc get** 命令以查看所创建的应用程序：

```
$ oc get application -n openshift-gitops
```

4. 在部署应用程序的命名空间中添加标签，以便 **openshift-gitops** 命名空间中的 Argo CD 实例可以管理它：

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

5.8.3. 验证 Argo CD 自助行为

Argo CD 持续监控已部署应用程序的状态，检测 Git 中指定清单和集群中的实时更改之间的差别，然后自动更正它们。这个行为被称为自我管理。

您可以在 Argo CD 中测试并观察自我管理的行为。

先决条件

- 已部署并配置 **app-spring-petclinic** 应用程序示例。

流程

1. 在 Argo CD 仪表板中，验证您的应用程序是否具有 **Synced** 状态。
2. 点 Argo CD 仪表板中的 **app-spring-petclinic** 标题，查看部署到集群中的应用程序资源。
3. 在 OpenShift Container Platform web 控制台中进入 **Developer** 视角。
4. 修改 Spring PetClinic 部署，并将更改提交到 Git 仓库的 **app/** 目录。Argo CD 将自动将更改部署到集群。
   a. Fork OpenShift GitOps getting started repository。
   b. 在 **deployment.yaml** 文件中，将 **failureThreshold** 值改为 **5**。

c. 在部署集群中，运行以下命令验证 **failureThreshold** 字段更改的值：

```
$ oc edit deployment spring-petclinic -n spring-petclinic
```

5. 通过修改集群上的部署并扩展到两个容器集来测试自我修复行为，同时在 OpenShift Container Platform Web 控制台中观察应用程序。

a. 运行以下命令修改部署：

```
$ oc scale deployment spring-petclinic --replicas 2  -n spring-petclinic
```

b. 在 OpenShift Container Platform Web 控制台中，请注意部署最多扩展两个 pod，并立即缩减到一个 pod。Argo CD 检测到与 Git 存储库的区别，并在 OpenShift Container Platform 集群中自动修复应用程序。

6. 在 Argo CD 仪表板中，点 **app-spring-petclinic** 标题 → **APP DETAILS** → **EVENTS**。**EVENTS** 选项卡显示以下事件： Argo CD 检测集群中缺少同步部署资源，然后重新同步 Git 存储库进行更正。

5.9. ARGO CD OPERATOR

ArgoCD 自定义资源(CRD)是一个 Kubernetes 自定义资源(CRD)，用于描述给定 Argo CD 集群的所需状态，允许您配置组成 Argo CD 集群的组件。

5.9.1. Argo CD CLI 工具

Argo CD CLI 工具用于通过命令行配置 Argo CD。Red Hat OpenShift GitOps 不支持这个二进制文件。使用 OpenShift Console 配置 Argo CD。

5.9.2. Argo CD 自定义资源属性

Argo CD 自定义资源由以下属性组成：

Name	描述	default	Properties
ApplicationInstanceLabelKey	Argo CD 注入应用程序名称的 **metadata.label** 键名称作为一个跟踪标签。	**app.kubernetes.io/instance**	

ApplicationSet	**ApplicationSet** 控制器配置选项。	***<Object>***	• *<Image>* - **ApplicationSet** 控制器的容器镜像。这会覆盖 **ARGOCD_APPLICATIONSET_IMAGE** 环境变量。 • *<Version>* - **ApplicationSet** 容器镜像使用的标签。 • *<Resources>* - 容器计算资源。 • *<LogLevel>* - Argo CD Application Controller 组件使用的日志级别。有效选项包括 **debug**、**info**、**error** 和 **warn**。 • *<LogFormat>* - Argo CD Application Controller 组件使用的日志格式。有效选项为 **text** 或 **json**。 • *<PrallelismLimit>* - 为控制器设置的 kubectl parallelism 限制 **(--kubectl-parallelism-limit 标志)**。
ConfigManagementPlugins	添加配置管理插件。	***<empty>***	

Controller	Argo CD Application Controller 选项。	***<Object>***	• *<Processors.Operation>* - 操作处理器数量。 • *<Processors.Status>* - 状态处理器数量。 • *<Resources>* - 容器计算资源。 • *<LogLevel>* - Argo CD Application Controller 组件使用的日志级别。有效选项包括 **debug**、**info**、**error** 和 **warn**。 • *<AppSync>* - AppSync 用于控制 Argo CD 应用程序的同步频率 • *<Sharding.enabled>* - 在 Argo CD Application Controller 组件中启用分片。此属性用于管理大量集群，以减轻控制器组件的内存压力。 • *<Sharding.replicas>* - 用于支持 Argo CD Application Controller 的分片的副本数量。 • *<Env>* - 为应用程序控制器工作负载设置的环境。
DisableAdmin	禁用内置的 admin 用户。	**false**	
GATrackingID	使用 Google Analytics 跟踪 ID。	***<empty>***	
GAAnonymizeusers	启用发送至 google 分析的散列用户名。	**false**	

HA	高可用性选项。	***<Object>***	• *<Enabled>* - 为 Argo CD 切换全局的高可用性支持。 • *<RedisProxyImage>* - Redis HAProxy 容器镜像。这会覆盖 **ARGOCD_REDIS_HA_PROXY_IMAGE** 环境变量。 • *<RedisProxyVersion>* - 用于 Redis HAProxy 容器镜像的标签。
HelpChatURL	用于获取聊天帮助的 URL（通常是您的 Slack 频道支持）。	**https://mycorp.slack.com/argo-cd**	
HelpChatText	显示在进行聊天帮助的文本框中。	**现在聊天！**	
Image	所有 Argo CD 组件的容器镜像。这会覆盖 **ARGOCD_IMAGE** 环境变量。	**argoproj/argocd**	
入口	Ingress 配置选项。	***<Object>***	
InitialRepositories	初始 Git 存储库，将 Argo CD 配置为在创建集群时使用。	***<empty>***	

通知	通知控制器配置选项。	***<Object>***	• *<Enabled>* - 切换启动 notifications-controller。 • *<Image>* - 所有 Argo CD 组件的容器镜像。这会覆盖 **ARGOCD_IMAGE** 环境变量。 • *<Version>* - 与 Notifications 容器镜像一起使用的标签。 • *<Resources>* - 容器计算资源。 • *<LogLevel>* - Argo CD Application Controller 组件使用的日志级别。有效选项包括 **debug**、**info**、**error** 和 **warn**。
RepositoryCredentials	Git 存储库凭证模板，将 Argo CD 配置为在创建集群时使用。	***<empty>***	
InitialSSHKnownHosts	创建集群时要使用的初始 SSH 已知问题供 Argo CD 使用。	***<default_Argo_CD_Known_Hosts>***	
KustomizeBuildOptions	用于 **kustomize build** 的构建选项和参数。	***<empty>***	
OIDCConfig	OIDC 配置作为 Dex 的替代方案。	***<empty>***	
NodePlacement	添加 **nodeSelector** 和 **tolerations**。	***<empty>***	

Prometheus	Prometheus 配置选项。	***<Object>***	• *<Enabled>* - 为 Argo CD 切换全局的 Prometheus 支持。 • *<Host>* - 用于 Ingress 或 Route 资源的主机名。 • *<Ingress>* - 为 Prometheus 切换 Ingress。 • *<Route>* - 路由配置选项。 • *<Size>* - Prometheus **StatefulSet** 的副本数。
RBAC	RBAC 配置选项。	***<Object>***	• *<DefaultPolicy>* - 在 **argocd-rbac-cm** 配置映射的 **policy.default** 属性。在授权 API 请求时，Argo CD 将回退到的默认角色的名称。 • *<Policy>* - **argocd-rbac-cm** 配置映射中的 **policy.csv** 属性。包含用户定义的 RBAC 策略和角色定义的 CSV 数据。 • *<Scopes>* - **argocd-rbac-cm** 配置映射中的 scopes 属性。控制在 RBAC 执行过程中要检查的 OIDC 范围（子范围除外）。

Redis	Redis 配置选项。	***<Object>***	• *<AutoTLS>* - 使用供应商创建 Redis 服务器的 TLS 证书 (one of: openshift)。目前仅适用于 OpenShift Container Platform。 • *<DisableTLSVerification>* - 定义 Redis 服务器是否应该使用严格的 TLS 验证进行访问。 • *<Image>* - Redis 的容器镜像。这会覆盖 **ARGOCD_REDIS_IMAGE** 环境变量。 • *<Resources>* - 容器计算资源。 • *<Version>* - 与 Redis 容器镜像一起使用的标签。
ResourceCustomizations	自定义资源行为。	***<empty>***	
ResourceExclusions	完全忽略整个资源组类别。	***<empty>***	
ResourceInclusions	配置要应用哪些资源组/kinds 的配置。	***<empty>***	
Server	Argo CD Server 配置选项。	***<Object>***	• *<Autoscale>* - 服务器自动扩展配置选项。 • *<ExtraCommandArgs>* - Operator 设置的现有参数列表。 • *<GRPC>* - GRPC 配置选项。 • *<Host>* - 用于 Ingress 或 Route 资源的主

			机名。 • *<Ingress>* - Argo CD 服务器组件的 Ingress 配置。 • *<insecure>* - 切换 Argo CD 服务器的 insecure 标志。 • *<Resources>* - 容器计算资源。 • *<Replicas>* - Argo CD 服务器的副本数量。必须大于或等于 **0**。如果启用了 **Autoscale**，则忽略 **Replicas**。 • *<Route>* - 路由配置选项。 • *<Service.Type>* - 用于服务资源的 **ServiceType**。 • *<Loglevel>* - Argo CD Server 组件使用的日志级别。有效选项包括 **debug**、**info**、**error** 和 **warn**。 • *<LogFormat>* - Argo CD Application Controller 组件使用的日志格式。有效选项为 **text** 或 **json**。 • *<Env>* - 为服务器工作负载设置的环境。

SSO	单点登录选项。	***<Object>***	• *<Image>* - Keycloak 的容器镜像。这会覆盖 **ARGOCD_KEYCLOAK_IMAGE** 环境变量。 • *<Keycloak>* - Keycloak SSO 供应商的配置选项。 • *<Dex>* - Dex SSO 供应商的配置选项。 • *<Provider>* - 配置单点登录的供应商名称。现在，支持的选项有 Dex 和 Keycloak。 • *<Resources>* - 容器计算资源。 • *<VerifyTLS>* - 与 Keycloak 服务通信时用于强制进行严格的 TLS 检查。 • *<Version>* - 与 Keycloak 容器镜像一起使用的标签。
StatusBadgeEnabled	启用应用程序状态徽标。	**true**	
TLS	TLS 配置选项。	***<Object>***	• *<CA.ConfigMapName>* - 包含 CA 证书的 **ConfigMap** 名称。 • *<CA.SecretName>* - 包含 CA 证书和密钥的 secret 名称。 • *<InitialCerts>* - **argocd-tls-certs-cm** 配置映射中的初始证书集合，以通过 HTTPS 连接 Git 存储库。

UserAnonyousEnabled	启用匿名用户访问。	**true**	
版本	用于所有 Argo CD 组件的容器镜像的标签。	最新的 Argo CD 版本	
横幅	添加 UI 横幅消息。	***<Object>***	• *<Banner.Content>* - 横幅消息内容（如果显示了横幅，则必需）。 • *<Banner.URL.SecretName>* - 横幅消息链接 URL （可选）。

5.9.3. 仓库服务器属性

以下属性可用于配置 Repo 服务器组件：

Name	**默认**	**描述**
Resources	***<empty>***	容器计算资源。
MountSAToken	**false**	**ServiceAccount** 令牌是否应挂载到 repo-server pod。
ServiceAccount	**""**	与 repo-server pod 搭配使用的 **ServiceAccount** 的名称。
VerifyTLS	**false**	在与仓库服务器通信时，是否在所有组件上执行严格的 TLS 检查。
AutoTLS	**""**	用于设置 TLS provider，用于设置 repo-server 的 gRPC TLS 证书（：openshift 之一）。目前仅适用于 OpenShift。
Image	**argoproj/argocd**	Argo CD Repo 服务器的容器镜像。这会覆盖 **ARGOCD_REPOSERVER_IMAGE** 环境变量。
版本	与 **.spec.Version** 相同	与 Argo CD Repo 服务器一起使用的标签。

LogLevel	**info**	Argo CD Repo 服务器使用的日志级别。有效选项包括 debug、info、error 和 warn。
LogFormat	**text**	Argo CD Repo 服务器使用的日志格式。有效选项为 text 或 json。
ExecTimeout	**180**	呈现工具（如 Helm、Kustomize）的执行超时（如 Helm、Kustomize）。
Env	***<empty>***	为仓库服务器工作负载设置的环境。
Replicas	***<empty>***	Argo CD Repo 服务器的副本数。必须大于或等于 **0**。

5.9.4. 使用 Argo CD 实例启用通知

要启用或禁用 Argo CD 通知控制器，在 Argo CD 自定义资源中设置参数。默认情况下禁用通知。要启用通知，在 **.yaml** 文件中将 **enabled** 参数设置为 **true** ：

流程

1. 将 **enabled** 参数设置为 **true** ：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  notifications:
    enabled: true
```

5.10. 配置与 REDIS 的安全通信

在 Red Hat OpenShift GitOps 中使用传输层安全 (TLS) 加密，您可以保护 Argo CD 组件和 Redis 缓存之间的通信，并保护传输中潜在的敏感数据。

您可以使用以下配置之一保护与 Redis 的通信：

- 启用 **autotls** 设置，为 TLS 加密发布适当的证书。
- 通过使用密钥和证书对创建 **argocd-operator-redis-tls** secret，手动配置 TLS 加密。

启用或没有启用高可用性 (HA) 时都可以使用这两个配置。

先决条件

- 您可以使用 **cluster-admin** 权限访问集群。

- 访问 OpenShift Container Platform web 控制台。
- 在集群中安装了 Red Hat OpenShift GitOps Operator。

5.10.1. 为启用了 autotls 的 Redis 配置 TLS

您可以通过在新的或已有的 Argo CD 实例中启用 **autotls** 设置来为 Redis 配置 TLS 加密。配置会自动置备 **argocd-operator-redis-tls** secret，且不需要进一步的步骤。目前，OpenShift Container Platform 是唯一受支持的 secret 供应商。

注意

默认情况下禁用 **autotls** 设置。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 创建启用了 **autotls** 的 Argo CD 实例：
   a. 在 Web 控制台的 **Administrator** 视角中，使用左侧导航面板进入 **Administration → CustomResourceDefinitions**。
   b. 搜索 **argocds.argoproj.io** 并点 **ArgoCD** 自定义资源定义 (CRD)。
   c. 在 **CustomResourceDefinition 详情**页面中，点 **Instances** 选项卡，然后点 **Create ArgoCD**。
   d. 编辑或替换类似以下示例的 YAML：

启用 autotls 的 Argo CD CR 示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  redis:
    autotls: openshift (3)
  ha:
    enabled: true (4)
```

(1) Argo CD 实例的名称。
(2) 要运行 Argo CD 实例的命名空间。
(3) 启用 **autotls** 设置并为 Redis 创建 TLS 证书的标记。
(4) 启用 HA 功能的 flag 值。如果不启用 HA，请不要包含此行，或者将标志值设为 **false**。

提示

另外，您可以通过运行以下命令来在已经存在的 Argo CD 实例上启用 **autotls** 设置：

```
$ oc patch argocds.argoproj.io <instance-name> --type=merge -p '{"spec":{"redis":
{"autotls":"openshift"}}}'
```

e. 点 **Create**。

f. 验证 Argo CD pod 是否已就绪并在运行：

```
$ oc get pods -n <namespace> 1
```

1 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

禁用 HA 的输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```

注意

启用 HA 的 TLS 配置需要一个至少有三个 worker 节点的集群。如果您启用了使用 HA 配置的 Argo CD 实例，可能需要几分钟时间才会显示输出。

启用了 HA 的输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h  1/1     Running   0          10m
argocd-redis-ha-server-0            2/2     Running   0          9m9s
argocd-redis-ha-server-1            2/2     Running   0          98s
argocd-redis-ha-server-2            2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh 1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks       1/1     Running   0          10m
```

3. 验证 **argocd-operator-redis-tls** secret 是否已创建：

```
$ oc get secrets argocd-operator-redis-tls -n <namespace> 1
```

1 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

输出示例

```
NAME                     TYPE                DATA   AGE
argocd-operator-redis-tls  kubernetes.io/tls   2      30s
```

secret 必须是 **kubernetes.io/tls** 类型，大小为 **2**。

5.10.2. 为禁用了 autotls 的 Redis 配置 TLS

您可以使用密钥和证书对创建 **argocd-operator-redis-tls** secret，为 Redis 手动配置 TLS 加密。另外，您必须注解 secret 以指示它属于适当的 Argo CD 实例。对于启用了高可用性 (HA) 的实例，创建证书和 secret 的步骤会有所不同。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。
2. 创建 Argo CD 实例：
   a. 在 Web 控制台的 **Administrator** 视角中，使用左侧导航面板进入 **Administration → CustomResourceDefinitions**。
   b. 搜索 **argocds.argoproj.io** 并点 **ArgoCD** 自定义资源定义 (CRD)。
   c. 在 **CustomResourceDefinition 详情**页面中，点 **Instances** 选项卡，然后点 **Create ArgoCD**。
   d. 编辑或替换类似以下示例的 YAML：

禁用 autotls 的 ArgoCD CR 示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  ha:
    enabled: true (3)
```

(1) Argo CD 实例的名称。
(2) 要运行 Argo CD 实例的命名空间。
(3) 启用 HA 功能的 flag 值。如果不启用 HA，请不要包含此行，或者将标志值设为 **false**。

   e. 点 **Create**。
   f. 验证 Argo CD pod 是否已就绪并在运行：

```
$ oc get pods -n <namespace> (1)
```

(1) 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

禁用 HA 的输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
```

```
argocd-redis-84b77d4f58-vp6zm         1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq   1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh        1/1     Running   0          37s
```

注意

启用 HA 的 TLS 配置需要一个至少有三个 worker 节点的集群。如果您启用了使用 HA 配置的 Argo CD 实例，可能需要几分钟时间才会显示输出。

启用了 HA 的输出示例

```
NAME                                          READY   STATUS    RESTARTS   AGE
argocd-application-controller-0               1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h      1/1     Running   0          10m
argocd-redis-ha-server-0                      2/2     Running   0          9m9s
argocd-redis-ha-server-1                      2/2     Running   0          98s
argocd-redis-ha-server-2                      2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh           1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks                 1/1     Running   0          10m
```

3. 根据您的 HA 配置，使用以下选项之一为 Redis 服务器创建一个自签名证书：

- 对于禁用了 HA 的 Argo CD 实例，请运行以下命令：

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
  -keyout /tmp/redis.key \
  -out /tmp/redis.crt \
  -newkey rsa:4096 \
  -nodes \
  -sha256 \
  -days 10
```

1 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

输出示例

```
Generating a RSA private key
..............++++
.............................++++
writing new private key to '/tmp/redis.key'
```

- 对于启用了 HA 的 Argo CD 实例，运行以下命令：

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis-ha-haproxy.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
```

```
-keyout /tmp/redis-ha.key \
-out /tmp/redis-ha.crt \
-newkey rsa:4096 \
-nodes \
-sha256 \
-days 10
```

1 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

输出示例

```
Generating a RSA private key
..............++++
.............................++++
writing new private key to '/tmp/redis-ha.key'
```

4. 运行以下命令，验证生成的证书和密钥是否在 **/tmp** 目录中可用：

```
$ cd /tmp
```

```
$ ls
```

禁用 HA 的输出示例

```
...
redis.crt
redis.key
...
```

启用了 HA 的输出示例

```
...
redis-ha.crt
redis-ha.key
...
```

5. 根据您的 HA 配置，使用以下选项之一创建 **argocd-operator-redis-tls** secret：

- 对于禁用了 HA 的 Argo CD 实例，请运行以下命令：

```
$ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis.key --cert=/tmp/redis.crt
```

- 对于启用了 HA 的 Argo CD 实例，运行以下命令：

```
$ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis-ha.key --cert=/tmp/redis-ha.crt
```

输出示例

```
secret/argocd-operator-redis-tls created
```

6. 注解 secret 以表示它属于 Argo CD CR：

```
$ oc annotate secret argocd-operator-redis-tls argocds.argoproj.io/name=<instance-name>
1
```

1 指定 Argo CD 实例的名称，如 **argocd**。

输出示例

```
secret/argocd-operator-redis-tls annotated
```

7. 验证 Argo CD pod 是否已就绪并在运行：

```
$ oc get pods -n <namespace> 1
```

1 指定运行 Argo CD 实例的命名空间，如 **openshift-gitops**。

禁用 HA 的输出示例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```

注意

如果您启用了使用 HA 配置的 Argo CD 实例，可能需要几分钟时间才会显示输出。

启用了 HA 的输出示例

```
NAME                                          READY   STATUS    RESTARTS   AGE
argocd-application-controller-0               1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h      1/1     Running   0          10m
argocd-redis-ha-server-0                      2/2     Running   0          9m9s
argocd-redis-ha-server-1                      2/2     Running   0          98s
argocd-redis-ha-server-2                      2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh           1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks                 1/1     Running   0          10m
```

5.11. 监控应用程序资源和部署的健康状况信息

OpenShift Container Platform Web 控制台的 **Developer** 视角中的 Red Hat OpenShift GitOps **Environments** 页面显示应用程序环境成功部署的列表，以及指向每个部署的修订版本的链接。

OpenShift Container Platform Web 控制台的 **Developer** 视角中的 **Application environments** 页面显示应用程序资源的健康状况，如路由、同步状态、部署配置和部署历史记录。

OpenShift Container Platform Web 控制台的 **Developer** 视角中的环境页面与 Red Hat OpenShift

GitOps Application Manager 命令行界面(CLI) **kam** 分离。您不必使用 **kam** 为环境生成应用程序环境清单，以便在 OpenShift Container Platform Web 控制台的 **Developer** 视角中显示。您可以使用自己的清单，但环境仍必须由命名空间表示。另外，仍然需要特定的标签和注解。

5.11.1. 环境标签和注解的设置

本节在 OpenShift Container Platform Web 控制台的 **Developer** 视角中提供在 **Environments** 页面中显示环境应用程序所需的环境标签和注解的引用设置。

环境标签
环境应用程序清单必须包含 **labels.openshift.gitops/environment** 和 **destination.namespace** 字段。您必须为 < **environment_name&gt**; 变量和环境应用程序清单的名称设置相同的值。

环境应用程序清单的规格

```
spec:
  labels:
    openshift.gitops/environment: <environment_name>
  destination:
    namespace: <environment_name>
...
```

环境应用程序清单示例

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: dev-env 1
  namespace: openshift-gitops
spec:
  labels:
    openshift.gitops/environment: dev-env
  destination:
    namespace: dev-env
...
```

1 环境应用清单的名称。设置的值与 < **environment_name&** gt; 变量的值相同。

环境注解
环境命名空间清单必须包含 **annotations.app.openshift.io/vcs-uri** 和 **annotations.app.openshift.io/vcs-ref** 字段来指定应用程序的版本控制器代码源。您必须为 < **environment_name&gt**; 变量和环境命名空间清单的名称设置相同的值。

环境命名空间清单的规格

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    app.openshift.io/vcs-uri: <application_source_url>
    app.openshift.io/vcs-ref: <branch_reference>
  name: <environment_name> 1
...
```

1 环境命名空间清单的名称。设置的值与 < **environment_name&** gt; 变量的值相同。

环境命名空间清单示例

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    app.openshift.io/vcs-uri: https://example.com/<your_domain>/<your_gitops.git>
    app.openshift.io/vcs-ref: main
  labels:
    argocd.argoproj.io/managed-by: openshift-gitops
  name: dev-env
...
```

5.11.2. 检查健康信息

Red Hat OpenShift GitOps Operator 将在 **openshift-gitops** 命名空间中安装 GitOps 后端服务。

先决条件

- Red Hat OpenShift GitOps Operator 从 **OperatorHub** 安装。
- 确保您的应用程序由 Argo CD 同步。

流程

1. 点 **Developer** 视角下的 **Environments**。**Environments** 页面中显示应用程序列表及其 **环境状态**。
2. 将鼠标悬停在 **Environment status** 列下的图标上，以查看所有环境的同步状态。
3. 点击列表中的应用程序名称查看特定应用程序的详情。
4. 在 **Application environments** 页面中，如果 **Overview** 选项卡下的 **Resources** 部分显示图标，将鼠标悬停在图标上来获取状态详情。
   - 一个分离开的心型图标表示资源问题已降低应用程序的性能。
   - 一个黄色的符号表示资源问题中带有应用程序的健康状态的延迟数据。

5.12. 使用 DEX 为 ARGO CD 配置 SSO

安装 Red Hat OpenShift GitOps Operator 后，Argo CD 会自动创建一个具有 **admin** 权限的用户。要管理多个用户，集群管理员可以使用 Argo CD 来配置 Single Sign-On(SSO)。

重要

ArgoCD CR 中的 **spec.dex** 参数已弃用。在以后的 Red Hat OpenShift GitOps v1.9 发行版本中，计划使用 ArgoCD CR 中的 **spec.dex** 参数配置 Dex。考虑改用 **.spec.sso** 参数。

5.12.1. 启用 Dex OpenShift OAuth 连接器

Dex 通过检查平台提供的 **OAuth** 服务器，使用 OpenShift 中定义的用户和组。以下示例显示了 Dex 的属性以及示例配置：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: openshift-oauth
spec:
  dex:
    openShiftOAuth: true 1
    groups: 2
     - default
  rbac: 3
    defaultPolicy: 'role:readonly'
    policy: |
      g, cluster-admins, role:admin
    scopes: '[groups]'
```

1. **openShiftOAuth** 属性触发 Operator，当值设为 **true** 时自动配置内置 OpenShift **OAuth** 服务器。
2. **groups** 属性允许指定组的用户登录。
3. RBAC 策略属性将 Argo CD 集群中的 admin 角色分配给 OpenShift **cluster-admins** 组中的用户。

5.12.1.1. 将用户映射到特定的角色

如果有直接 **ClusterRoleBinding** 角色，Argo CD 无法将用户映射到特定角色。您可以通过 OpenShift，手动更改 SSO 上的 **role:admin** 角色。

流程

1. 创建名为 **cluster-admins** 的组。

```
$ oc adm groups new cluster-admins
```

2. 将用户添加到组。

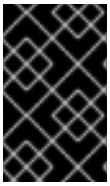
```
$ oc adm groups add-users cluster-admins USER
```

3. 将 **cluster-admin ClusterRole** 应用到组：

```
$ oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
```

5.12.2. 禁用 Dex

对于 Operator 创建的所有 Argo CD 实例，默认安装 Dex。您可以通过设置 **.spec.dex** 参数，将 Red Hat OpenShift GitOps 配置为使用 Dex 作为 SSO 身份验证提供程序。



重要

在 Red Hat OpenShift GitOps v1.6.0 中，**DISABLE_DEX** 已被弃用，计划在 Red Hat OpenShift GitOps v1.9.0 中删除。请考虑使用 **.spec.sso.dex** 参数。请参阅 "使用 .spec.sso 启用或禁用 Dex"。

流程

- 在 Operator 的 **YAML** 资源中将环境变量 **DISABLE_DEX** 设置为 true：

```
...
spec:
  config:
    env:
    - name: DISABLE_DEX
      value: "true"
...
```

5.12.3. 使用 .spec.sso 启用或禁用 Dex

您可以通过设置 **.spec.sso** 参数，将 Red Hat OpenShift GitOps 配置为使用 Dex 作为其 SSO 身份验证供应商。

流程

1. 要启用 Dex，请在 Operator 的 YAML 资源中设置 **.spec.sso.provider: dex** 参数：

```
...
spec:
  sso:
    provider: dex
    dex:
      openShiftOAuth: true
...
```

2. 要禁用 dex，可以从 Argo CD 自定义资源中删除 **spec.sso** 元素，或指定不同的 SSO 供应商。

5.13. 使用 KEYCLOAK 为 ARGO CD 配置 SSO

安装 Red Hat OpenShift GitOps Operator 后，Argo CD 会自动创建一个具有 **admin** 权限的用户。要管理多个用户，集群管理员可以使用 Argo CD 来配置 Single Sign-On(SSO)。

先决条件

- 在集群中安装了 Red Hat SSO。
- 在集群中安装了 Red Hat OpenShift GitOps Operator。
- 在集群中安装了 Argo CD。

5.13.1. 在 Keycloak 中配置新客户端

对于 Operator 创建的所有 Argo CD 实例，默认安装 Dex。但是，您可以删除 Dex 配置并添加 Keycloak，以使用 OpenShift 凭证登录到 Argo CD。Keycloak 作为 Argo CD 和 OpenShift 之间的身份代理。

流程

要配置 Keycloak，请按照以下步骤执行：

1. 通过从 Argo CD 自定义资源 (CR) 中删除 **.spec.sso.dex** 参数来删除 Dex 配置，并保存 CR：

```
dex:
  openShiftOAuth: true
  resources:
    limits:
      cpu:
      memory:
    requests:
      cpu:
      memory:
```

2. 在 Argo CD CR 中将 **provider** 参数的值设置为 **keycloak**。

3. 通过执行以下步骤配置 Keycloak：

   - 对于安全连接，设置 **rootCA** 参数的值，如下例所示：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: "<PEM-encoded-root-certificate>" 1
  server:
    route:
      enabled: true
```

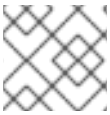
   1 用于验证 Keycloak 的 TLS 证书的自定义证书。

   Operator 会协调 **.spec.keycloak.rootCA** 参数中的更改，并使用 **argocd-cm** 配置映射中的 PEM 编码 root 证书更新 **oidc.config** 参数。

   - 对于不安全连接，将 **rootCA** 参数的值留空，并使用 **oidc.tls.insecure.skip.verify** 参数，如下所示：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
```

```
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
  sso:
    provider: keycloak
    keycloak:
      rootCA: ""
```



注意

Keycloak 实例需要 2-3 分钟来安装和运行。

5.13.2. 登录到 Keycloak

登录到 Keycloak 控制台以管理身份或角色，并定义分配给不同角色的权限。

先决条件

- 删除 Dex 的默认配置。
- Argo CD CR 必须配置为使用 Keycloak SSO 供应商。

流程

1. 获取用于登录的 Keycloak 路由 URL：

```
$ oc -n argocd get route keycloak

NAME       HOST/PORT                                              PATH   SERVICES   PORT
TERMINATION   WILDCARD
keycloak   keycloak-default.apps.ci-ln-******.origin-ci-int-aws.dev.**.com          keycloak   <all>
reencrypt     None
```

2. 获取将用户名和密码存储为环境变量的 Keycloak pod 名称：

```
$ oc -n argocd get pods

NAME                  READY   STATUS          RESTARTS   AGE
keycloak-1-2sjcl          1/1     Running           0        45m
```

   a. 获取 Keycloak 用户名：

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_USERNAME

SSO_ADMIN_USERNAME=<username>
```

   b. 获取 Keycloak 密码：

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_PASSWORD

SSO_ADMIN_PASSWORD=<password>
```

3. 在登录页面上，点 **LOG IN VIA KEYCLOAK**。

注意

您只能在 Keycloak 实例就绪后看到 **LOGIN VIA KEYCLOAK** 选项。

4. 点 **Login with OpenShift**。

注意

不支持使用 **kubeadmin** 登录。

5. 输入要登录的 OpenShift 凭据。

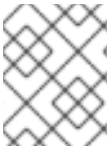
6. 可选：默认情况下，登录到 Argo CD 的任何用户都具有只读访问权限。您可以通过更新 **argocd-rbac-cm** 配置映射来管理用户级别访问权限：

```
policy.csv:
<name>, <email>, role:admin
```

5.13.3. 卸载 Keycloak

您可以通过从 Argo CD 自定义资源(CR)文件中删除 **SSO** 字段来删除 Keycloak 资源及其相关配置。删除 **SSO** 字段后，文件中的值类似如下：

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  server:
    route:
     enabled: true
```



注意

使用此方法创建的 Keycloak 应用程序当前不是持久性。在服务器重启时，在 Argo CD Keycloak 域中创建的其他配置会被删除。

5.14. 配置 ARGO CD RBAC

默认情况下，如果您使用 RHSSO 登录到 Argo CD，则是一个只读用户。您可以更改并管理用户级别访问权限。

5.14.1. 配置用户级别访问权限

要管理和修改用户级别访问权限，请在 Argo CD 自定义资源中配置 RBAC 部分。

流程

- 编辑 **argocd** 自定义资源：

```
$ oc edit argocd [argocd-instance-name] -n [namespace]
```

输出

```
metadata
...
...
  rbac:
    policy: 'g, rbacsystem:cluster-admins, role:admin'
    scopes: '[groups]'
```

- 将**策略**配置添加到 **rbac** 部分，再添加用户的 **name**, **email** 和 **role**。

```
metadata
...
...
rbac:
    policy: <name>, <email>, role:<admin>
    scopes: '[groups]'
```

注意

目前，RHSSO 无法读取 Red Hat OpenShift GitOps 用户的组信息。因此，在用户级别配置 RBAC。

5.14.2. 修改 RHSSO 资源请求/限制

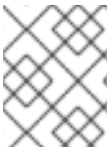
默认情况下，RHSSO 容器创建有资源请求和限值。您可以更改并管理资源请求。

资源	Requests	Limits
CPU	500	1000m
内存	512 Mi	1024 Mi

流程

修改默认资源要求，修补 Argo CD CR：

```
$ oc -n openshift-gitops patch argocd openshift-gitops --type='json' -p='[{"op": "add", "path": "/spec/sso", "value": {"provider": "keycloak", "resources": {"requests": {"cpu": "512m", "memory": "512Mi"}, "limits": {"cpu": "1024m", "memory": "1024Mi"}} }}]'
```



注意

Red Hat OpenShift GitOps 创建的 RHSSO 仅保留操作器所进行的更改。如果 RHSSO 重新启动，则 RHSSO 中的 Admin 创建的额外配置也会被删除。

5.15. 配置资源配额或请求

使用 Argo CD 自定义资源，您可以为 Argo CD 工作负载创建、更新和删除资源请求和限值。

5.15.1. 使用资源请求和限值配置工作负载

您可以使用资源请求和限值创建 Argo CD 自定义资源工作负载。当您要在配置了资源配额的命名空间中部署 Argo CD 实例时，这是必需的。

以下 Argo CD 实例部署 Argo CD 工作负载，如 **Application Controller**、**ApplicationSet Controller**、**Dex**、**Redis**、**Repo Server**, 和 **Server**，以及带有资源请求和限值的 Server。您还可以以同样的方式创建具有资源要求的其他工作负载。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example
spec:
  server:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 125m
        memory: 128Mi
    route:
      enabled: true
  applicationSet:
    resources:
      limits:
        cpu: '2'
        memory: 1Gi
      requests:
        cpu: 250m
        memory: 512Mi
  repo:
    resources:
      limits:
        cpu: '1'
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  dex:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 250m
        memory: 128Mi
  redis:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
```

```
      cpu: 250m
      memory: 128Mi
  controller:
    resources:
      limits:
        cpu: '2'
        memory: 2Gi
      requests:
        cpu: 250m
        memory: 1Gi
```

5.15.2. 修补 Argo CD 实例以更新资源要求

您可在安装后更新所有或任何工作负载的资源要求。

流程

更新 Argo CD 命名空间中的 Argo CD 实例的 **Application Controller** 资源请求。

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/cpu", "value":"1"}]'

oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/memory", "value":"512Mi"}]'
```

5.15.3. 删除资源请求

您还可以在安装后删除所有或任何工作负载的资源要求。

流程

删除 Argo CD 命名空间中的 Argo CD 实例的 **Application Controller** 资源请求。

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/cpu"}]'

oc -n argocd argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/memory"}]'
```

5.16. 监控 ARGO CD 自定义资源工作负载

使用 Red Hat OpenShift GitOps，您可以监控特定 Argo CD 实例的 Argo CD 自定义资源工作负载的可用性。通过监控 Argo CD 自定义资源工作负载，您可以通过为它们启用警报来获取有关 Argo CD 实例状态的最新信息。当相应 Argo CD 实例的组件工作负载 pod （如 application-controller、repo-server 或服务器）无法正常分配，且就绪副本数和所需副本数之间有偏差时，Operator 会触发警报。

您可以启用和禁用用于监控 Argo CD 自定义资源工作负载的设置。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 在集群中安装了 Red Hat OpenShift GitOps。

- 监控堆栈在 **openshift-monitoring** 项目中配置。另外，Argo CD 实例位于您可以通过 Prometheus 监控的命名空间中。
- **kube-state-metrics** 服务在集群中运行。
- 可选：如果您要为用户定义的项目中已存在 Argo CD 实例的监控，请确保为集群中的用户定义的项目启用了监控。

注意

如果要为默认 **openshift-monitoring** 堆栈监视的命名空间中的 Argo CD 实例启用监控，例如，任何不是以 **openshiftbang** 开头的命名空间，您必须在集群中启用用户工作负载监控。此操作可让监控堆栈获取创建的 PrometheusRule。

5.16.1. 为 Argo CD 自定义资源工作负载启用监控

默认情况下，Argo CD 自定义资源工作负载的监控配置被设置为 **false**。

在 Red Hat OpenShift GitOps 中，您可以为特定的 Argo CD 实例启用工作负载监控。因此，Operator 会创建一个 **PrometheusRule** 对象，其中包含由特定 Argo CD 实例管理的所有工作负载的警报规则。当相应组件的副本数从特定时间达到所需状态时，这些警报规则会触发警报。Operator 不会覆盖用户对 **PrometheusRule** 对象所做的更改。

流程

1. 在给定的 Argo CD 实例中，将 **.spec.monitoring.enabled** 字段值设置为 **true** ：

   Argo CD 自定义资源示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  ...
  monitoring:
    enabled: true
  ...
```

2. 验证 Operator 创建的 PrometheusRule 中是否包含警报规则：

   警报规则示例

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: argocd-component-status-alert
  namespace: openshift-gitops
spec:
  groups:
    - name: ArgoCDComponentStatus
      rules:
        ...
```

```
      - alert: ApplicationSetControllerNotReady (1)
        annotations:
          message: >-
            applicationSet controller deployment for Argo CD instance in
            namespace "default" is not running
        expr: >-
          kube_statefulset_status_replicas{statefulset="openshift-gitops-application-controller
statefulset",
          namespace="openshift-gitops"} !=
          kube_statefulset_status_replicas_ready{statefulset="openshift-gitops-application-
controller statefulset",
          namespace="openshift-gitops"}
        for: 1m
        labels:
          severity: critical
```

(1) PrometheusRule 中的警报规则，用于检查 Argo CD 实例创建的工作负载是否如预期运行。

5.16.2. 禁用 Argo CD 自定义资源工作负载的监控

您可以为特定的 Argo CD 实例禁用工作负载监控。禁用工作负载监控会删除创建的 PrometheusRule。

流程

- 在给定的 Argo CD 实例中，将 **.spec.monitoring.enabled** 字段值设置为 **false** ：

  Argo CD 自定义资源示例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  ...
  monitoring:
    enabled: false
  ...
```

5.16.3. 其他资源

- 为用户定义的项目启用监控

5.17. 查看 ARGO CD 日志

您可以使用 Red Hat OpenShift 的 logging 子系统查看 Argo CD 日志。logging 子系统可视觉化 Kibana 仪表板上的日志。OpenShift Logging Operator 默认启用 Argo CD 的日志记录。

5.17.1. 存储和检索 Argo CD 日志

您可以使用 Kibana 仪表板存储和检索 Argo CD 日志。

先决条件

- 在集群中安装了 Red Hat OpenShift GitOps Operator。
- Red Hat OpenShift 的 logging 子系统安装有集群中的默认配置。

流程

1. 在 OpenShift Container Platform web 控制台中，进入  菜单 → **Observability** → **Logging** 来查看 Kibana 仪表板。
2. 创建索引模式。
   a. 要显示所有索引，请将索引模式定义为 *，然后点 **Next step**。
   b. 为 **Time Filter field name** 选择 **@timestamp**。
   c. 单击 **Create index pattern**。
3. 在 Kibana 仪表板的导航面板中，点 **Discover** 选项卡。
4. 创建过滤器以检索 Argo CD 的日志。以下步骤创建一个过滤器，用于检索 **openshift-gitops** 命名空间中的所有 pod 的日志：
   a. 点 **Add a filter +**。
   b. 选择 **kubernetes.namespace_name** 字段。
   c. 选择 **is** operator。
   d. 选择 **openshift-gitops** 值。
   e. 点击 **Save**。
5. 可选：添加额外的过滤器来缩小搜索范围。例如，要检索特定 pod 的日志，您可以使用 **kubernetes.pod_name** 作为字段创建另一个过滤器。
6. 在 Kibana 仪表板中查看过滤的 Argo CD 日志。

5.17.2. 其他资源

- 使用 Web 控制台为 Red Hat OpenShift 安装 logging 子系统

5.18. 在基础架构节点上运行 GITOPS CONTROL PLANE 工作负载

您可以使用基础架构节点防止额外的账单成本。

您可以使用 OpenShift Container Platform 在 Red Hat OpenShift GitOps Operator 安装的基础架构节点上运行特定的工作负载。默认情况下，它由 Red Hat OpenShift GitOps Operator 安装的工作负载包括在 **openshift-gitops** 命名空间中，包括该命名空间中的默认 Argo CD 实例。

注意

任何安装到用户命名空间的其他 Argo CD 实例都无权在 Infrastructure 节点上运行。

5.18.1. 将 GitOps 工作负载移到基础架构节点

您可以将 Red Hat OpenShift GitOps 安装的默认工作负载移到基础架构节点。可移动的工作负载有：

- **kam deployment**
- **集群部署**（后端服务）
- **openshift-gitops-applicationset-controller 部署**
- **openshift-gitops-dex-server 部署**
- **openshift-gitops-redis 部署**
- **openshift-gitops-redis-ha-haproxy 部署**
- **openshift-gitops-repo-sever 部署**
- **openshift-gitops-server 部署**
- **openshift-gitops-application-controller statefulset**
- **openshift-gitops-redis-server statefulset**

流程

1. 运行以下命令，将现有节点标记为基础架构：

```
$ oc label node <node-name> node-role.kubernetes.io/infra=
```

2. 编辑 **GitOpsService** 自定义资源(CR)以添加基础架构节点选择器：

```
$ oc edit gitopsservice -n openshift-gitops
```

3. 在 **GitOpsService** CR 文件中，将 **runOnInfra** 字段添加到 **spec** 部分，并将其设置为 **true**。此字段将 **openshift-gitops** 命名空间中的工作负载移到基础架构节点：

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. 可选：在基础架构节点上应用污点并隔离工作负载，并防止其他工作负载调度到这些节点上。

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra
infra=reserved:NoSchedule infra=reserved:NoExecute
```

5. 可选： 如果您将污点应用到节点，您可以在 **GitOpsService** CR 中添加容限：

```
spec:
  runOnInfra: true
  tolerations:
  - effect: NoSchedule
```

```
      key: infra
      value: reserved
    - effect: NoExecute
      key: infra
      value: reserved
```

要验证工作负载是否已调度到 Red Hat OpenShift GitOps 命名空间中的基础架构节点上，请点击任何 pod 名称，并确保已添加了 **Node selector** 和 **Tolerations**。

注意

在默认 Argo CD CR 中手动添加节点选择器和 **Tolerations** 都会被 **GitOpsService** CR 中的切换和容限覆盖。

5.18.2. 其他资源

- 如需有关污点和容限的更多信息，请参阅使用节点污点控制 pod 放置。
- 有关基础架构机器集的更多信息，请参阅创建基础架构机器集。

5.19. GITOPS OPERATOR 的大小要求

大小要求页面显示在 OpenShift Container Platform 上安装 Red Hat OpenShift GitOps 的大小要求。它还提供由 GitOps Operator 实例化的默认 ArgoCD 实例的大小详情。

5.19.1. GitOps 的大小要求

Red Hat OpenShift GitOps 是为云原生应用程序实施持续部署的一种声明方法。通过 GitOps，您可以定义并配置应用程序的 CPU 和内存要求。

每次安装 Red Hat OpenShift GitOps Operator 时，命名空间上的资源都会在定义的限制中安装。如果默认安装没有设置任何限制或请求，Operator 会使用配额在命名空间中失败。如果没有足够资源，集群无法调度 ArgoCD 相关 pod。下表列出了默认工作负载的资源请求和限值：

Workload	CPU 请求	CPU 限值	内存请求	内存限值
argocd-application-controller	1	2	1024M	2048M
applicationset-controller	1	2	512M	1024M
argocd-server	0.125	0.5	128M	256M
argocd-repo-server	0.5	1	256M	1024M
argocd-redis	0.25	0.5	128M	256M
argocd-dex	0.25	0.5	128M	256M

Workload	CPU 请求	CPU 限值	内存请求	内存限值
HAProxy	0.25	0.5	128M	256M

另外，您还可以在 **oc** 命令中使用 ArgoCD 自定义资源来查看特定并修改它们：

```
oc edit argocd <name of argo cd> -n namespace
```

5.20. 对 RED HAT OPENSHIFT GITOPS 中的问题进行故障排除

在使用 Red Hat OpenShift GitOps 时，您可能会遇到与性能、监控、配置和其他方面相关的问题。本节帮助您了解这些问题并提供解决问题的解决方案。

5.20.1. 问题：在 Argo CD 与机器配置同步过程中自动重启

在 Red Hat OpenShift Container Platform 中，节点通过 Red Hat OpenShift Machine Config Operator (MCO) 自动更新。Machine Config Operator (MCO) 是一个自定义资源，供集群用来管理其节点的完整生命周期。

当在集群中创建或更新 MCO 资源时，MCO 会选择更新，对所选节点执行必要的更改，并通过封锁、排空和重新引导这些节点来安全地重启节点。它会处理从内核到 kubelet 的所有活动。

但是，MCO 和 GitOps 工作流之间的交互可能会带来重大性能问题和其他不必要的行为。本节介绍如何使 MCO 和 Argo CD GitOps 编配工具正常工作。

5.20.1.1. 解决方案 ：提高机器配置和 Argo CD 的性能

当您将 Machine Config Operator 用作 GitOps 工作流的一部分时，以下序列会生成子优化的性能：

- Argo CD 在提交包含应用程序资源的 Git 存储库后启动自动同步任务。
- 如果 Argo CD 在同步操作过程中发现新的或更新的机器配置，MCO 会获取机器配置的更改并开始重启节点以应用更改。
- 如果集群中的重新引导节点包含 Argo CD 应用程序控制器，应用程序控制器会终止，应用程序同步将中止。

当 MCO 按顺序重启节点时，可以在每次重启时重新调度 Argo CD 工作负载，可能需要一些时间才能完成同步。这会导致未定义的行为，直到 MCO 重启受同步中机器配置影响的所有节点。

5.20.2. 其他资源

- 防止节点在 Argo CD 与机器配置同步过程中自动重启