Red Hat OpenShift Pipelines 1.14

CI/CD パイプラインの作成

OpenShift Pipelines でのタスクとパイプラインの作成と実行の開始

Last Updated: 2024-03-27

Red Hat OpenShift Pipelines 1.14 CI/CD パイプラインの作成

OpenShift Pipelines でのタスクとパイプラインの作成と実行の開始

法律上の通知

概要

このドキュメントでは、OpenShift Pipelines でのタスクとパイプラインの作成と実行に関する情報を提供します。

目次

第1章 OPENSHIFT PIPELINES を使用したアプリケーションの CI/CD ソリューションの作成

Red Hat OpenShift Pipelines を使用すると、カスタマイズされた CI/CD ソリューションを作成して、アプリケーションをビルドし、テストし、デプロイできます。

アプリケーション向けの本格的なセルフサービス型の CI/CD パイプラインを作成するには、以下のタスクを実行する必要があります。

- カスタムタスクを作成するか、既存の再利用可能なタスクをインストールします。
- アプリケーションの配信パイプラインを作成し、定義します。
- 以下の方法のいずれかを使用して、パイプライン実行のためにワークスペースに接続されているストレージボリュームまたはファイルシステムを提供します。
  - 永続ボリューム要求 (PVC) を作成するボリューム要求テンプレートを指定します。
  - 永続ボリューム要求 (PVC) を指定します。
- **PipelineRun** オブジェクトを作成し、Pipeline をインスタンス化し、これを起動します。
- トリガーを追加し、ソースリポジトリーのイベントを取得します。

このセクションでは、**pipelines-tutorial** の例を使用して前述のタスクについて説明します。この例では、以下で設定される単純なアプリケーションを使用します。

- **pipelines-vote-ui** Git リポジトリーにソースコードがあるフロントエンドインターフェイス (**pipelines-vote-ui**)。
- **pipelines-vote-api** Git リポジトリーにソースコードがあるバックエンドインターフェイス (**pipelines-vote-api**)。
- **pipelines-tutorial** Git リポジトリーにある **apply-manifests** および **update-deployment** タスク。

1.1. 前提条件

- OpenShift Container Platform クラスターにアクセスできる。
- OpenShift OperatorHub にリストされている Red Hat OpenShift Pipelines Operator を使用して OpenShift Pipelines をインストールしている。インストールの完了後にクラスター全体に適用できる。
- OpenShift Pipelines CLI がインストールされている。
- GitHub ID を使用してフロントエンドの **pipelines-vote-ui** およびバックエンドの **pipelines-vote-api** Git リポジトリーをフォークしており、これらのリポジトリーに管理者権限でアクセスできる。
- オプション: **pipelines-tutorial** Git リポジトリーのクローンを作成している。

1.2. プロジェクトの作成およびパイプラインのサービスアカウントの確認

手順

1. OpenShift Container Platform クラスターにログインします。

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. サンプルアプリケーションのプロジェクトを作成します。このサンプルワークフローでは、**pipelines-tutorial** プロジェクトを作成します。

```
$ oc new-project pipelines-tutorial
```

注記

別の名前でプロジェクトを作成する場合は、サンプルで使用されているリソース URL をプロジェクト名で更新してください。

3. **pipeline** サービスアカウントを表示します。
Red Hat OpenShift Pipelines Operator は、イメージのビルドおよびプッシュを実行するのに十分なパーミッションを持つ **pipeline** という名前のサービスアカウントを追加し、設定します。このサービスアカウントは **PipelineRun** オブジェクトによって使用されます。

```
$ oc get serviceaccount pipeline
```

1.3. パイプラインタスクの作成

手順

1. **pipelines-tutorial** リポジトリーから **apply-manifests** および **update-deployment** タスクリソースをインストールします。これには、パイプラインの再利用可能なタスクのリストが含まれます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/01_pipeline/02_update_deployment_task.yaml
```

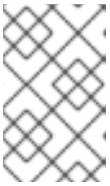
2. **tkn task list** コマンドを使用して、作成したタスクをリスト表示します。

```
$ tkn task list
```

出力では、**apply-manifests** および **update-deployment** タスクリソースが作成されていることを検証します。

```
NAME                DESCRIPTION   AGE
apply-manifests                   1 minute ago
update-deployment                 48 seconds ago
```

3. **tkn clustertasks list** コマンドを使用して、**buildah** および **s2i-python-3** などの Operator でインストールされた追加のクラスタータスクをリスト表示します。



注記

制限された環境で **buildah** クラスタータスクを使用するには、Dockerfile が内部イメージストリームをベースイメージとして使用していることを確認する必要があります。

```
$ tkn clustertasks list
```

出力には、Operator でインストールされた **ClusterTask** リソースが一覧表示されます。

```
NAME                DESCRIPTION   AGE
buildah                           1 day ago
git-clone                         1 day ago
s2i-python                         1 day ago
tkn                              1 day ago
```

重要

Red Hat OpenShift Pipelines 1.10 では、クラスタータスク機能は非推奨であり、将来のリリースで削除される予定です。

関連情報

- バージョン付けされていないクラスタータスクおよびバージョン付けされたクラスタータスクの管理

1.4. パイプラインのアセンブル

パイプラインは CI/CD フローを表し、実行するタスクによって定義されます。これは、複数のアプリケーションや環境で汎用的かつ再利用可能になるように設計されています。

パイプラインは、**from** および **runAfter** パラメーターを使用してタスクが相互に対話する方法および実行順序を指定します。これは **workspaces** フィールドを使用して、パイプラインの各タスクの実行中に必要な 1 つ以上のボリュームを指定します。

このセクションでは、GitHub からアプリケーションのソースコードを取り、これを OpenShift Container Platform にビルドし、デプロイするパイプラインを作成します。

パイプラインは、バックエンドアプリケーションの **vote-api** およびフロントエンドアプリケーション **vote-ui** について以下のタスクを実行します。

- **git-url** および **git-revision** パラメーターを参照して、Git リポジトリーからアプリケーションのソースコードのクローンを作成します。
- **buildah** クラスタータスクを使用してコンテナーイメージをビルドします。
- **image** パラメーターを参照して、イメージを OpenShift イメージレジストリーにプッシュします。
- **apply-manifests** および **update-deployment** タスクを使用して新規イメージを OpenShift Container Platform にデプロイします。

手順

1. 以下のサンプルのパイプライン YAML ファイルの内容をコピーし、保存します。

```
apiVersion: tekton.dev/v1
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-1.14"
  - 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
```

パイプライン定義は、Git ソースリポジトリーおよびイメージレジストリーの詳細を抽象化します。これらの詳細は、パイプラインのトリガーおよび実行時に **params** として追加されます。

2. パイプラインを作成します。

```
$ oc create -f <pipeline-yaml-file-name.yaml>
```

または、Git リポジトリーから YAML ファイルを直接実行することもできます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/01_pipeline/04_pipeline.yaml
```

3. **tkn pipeline list** コマンドを使用して、パイプラインがアプリケーションに追加されていることを確認します。

```
$ tkn pipeline list
```

この出力では、**build-and-deploy** パイプラインが作成されていることを検証します。

```
NAME              AGE            LAST RUN   STARTED   DURATION   STATUS
build-and-deploy  1 minute ago   ---        ---       ---        ---
```

1.5. 制限された環境でパイプラインを実行するためのイメージのミラーリング

OpenShift Pipelines を非接続のクラスターまたは制限された環境でプロビジョニングされたクラスターで実行するには、制限されたネットワークに Samples Operator が設定されているか、クラスター管理者がミラーリングされたレジストリーでクラスターを作成しているか確認する必要があります。

以下の手順では、**pipelines-tutorial** の例を使用して、ミラーリングされたレジストリーを持つクラスターを使用して、制限された環境でアプリケーションのパイプラインを作成します。**pipelines-tutorial** の例が制限された環境で機能することを確認するには、フロントエンドインターフェイス (**pipelines-vote-ui**)、バックエンドインターフェイス (**pipelines-vote-api**) および **cli** のミラーレジストリーからそれぞれのビルダーイメージをミラーリングする必要があります。

手順

1. フロントエンドインターフェイス (**pipelines-vote-ui**) のミラーレジストリーからビルダーイメージをミラーリングします。

   a. 必要なイメージタグがインポートされていないことを確認します。

   ```
   $ oc describe imagestream python -n openshift
   ```

   出力例

   ```
   Name:   python
   Namespace:  openshift
   [...]

   3.8-ubi9 (latest)
     tagged from registry.redhat.io/ubi9/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. サポートされるイメージタグをプライベートレジストリーに対してミラーリングします。

   ```
   $ oc image mirror registry.redhat.io/ubi9/python-39:latest <mirror-registry>:
   <port>/ubi9/python-39
   ```

   c. イメージをインポートします。

   ```
   $ oc tag <mirror-registry>:<port>/ubi9/python-39 python:latest --scheduled -n openshift
   ```

   イメージを定期的に再インポートする必要があります。**--scheduled** フラグは、イメージの自動再インポートを有効にします。

   d. 指定されたタグを持つイメージがインポートされていることを確認します。

   ```
   $ oc describe imagestream python -n openshift
   ```

   出力例

   ```
   Name:   python
   Namespace:  openshift
   [...]

   latest
     updates automatically from registry  <mirror-registry>:<port>/ubi9/python-39

     *  <mirror-registry>:<port>/ubi9/python-39@sha256:3ee...

   [...]
   ```

2. バックエンドインターフェイス (**pipelines-vote-api**) のミラーレジストリーからビルダーイメージをミラーリングします。

   a. 必要なイメージタグがインポートされていないことを確認します。

   ```
   $ oc describe imagestream golang -n openshift
   ```

   出力例

   ```
   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. サポートされるイメージタグをプライベートレジストリーに対してミラーリングします。

   ```
   $ oc image mirror registry.redhat.io/ubi9/go-toolset:latest <mirror-registry>:<port>/ubi9/go-toolset
   ```

   c. イメージをインポートします。

   ```
   $ oc tag <mirror-registry>:<port>/ubi9/go-toolset golang:latest --scheduled -n openshift
   ```

   イメージを定期的に再インポートする必要があります。**--scheduled** フラグは、イメージの自動再インポートを有効にします。

   d. 指定されたタグを持つイメージがインポートされていることを確認します。

   ```
   $ oc describe imagestream golang -n openshift
   ```

   出力例

   ```
   Name:   golang
   Namespace:  openshift
   [...]

   latest
     updates automatically from registry <mirror-registry>:<port>/ubi9/go-toolset

     * <mirror-registry>:<port>/ubi9/go-toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421
   ```

```
b37

[...]
```

3. **cli** のミラーレジストリーからビルダーイメージをミラーリングします。

   a. 必要なイメージタグがインポートされていないことを確認します。

   ```
   $ oc describe imagestream cli -n openshift
   ```

   出力例

   ```
   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. サポートされるイメージタグをプライベートレジストリーに対してミラーリングします。

   ```
   $ 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. イメージをインポートします。

   ```
   $ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest --
   scheduled -n openshift
   ```

   イメージを定期的に再インポートする必要があります。**--scheduled** フラグは、イメージの自動再インポートを有効にします。

   d. 指定されたタグを持つイメージがインポートされていることを確認します。

   ```
   $ oc describe imagestream cli -n openshift
   ```

   出力例

   ```
   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


[...]
```

関連情報

- 制限されたクラスターの Samples Operator の設定
- ミラーリングされたレジストリーでのクラスターの作成

1.6. パイプラインの実行

PipelineRun リソースはパイプラインを開始し、これを特定の呼び出しに使用する必要のある Git およびイメージリソースに関連付けます。これは、パイプラインの各タスクについて **TaskRun** を自動的に作成し、開始します。

手順

1. バックエンドアプリケーションのパイプラインを起動します。

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.14/01_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
```

直前のコマンドは、パイプライン実行の永続ボリューム要求 (PVC) を作成するボリューム要求テンプレートを使用します。

2. パイプライン実行の進捗を追跡するには、以下のコマンドを入力します。

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上記のコマンドの <pipelinerun_id> は、直前のコマンドの出力で返された **PipelineRun** の ID です。

3. フロントエンドアプリケーションのパイプラインを起動します。

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.14/01_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/pipelines-tutorial/pipelines-
vote-ui' \
    --use-param-defaults
```

4. パイプライン実行の進捗を追跡するには、以下のコマンドを入力します。

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

上記のコマンドの <pipelinerun_id> は、直前のコマンドの出力で返された **PipelineRun** の ID です。

5. 数分後に、**tkn pipelinerun list** コマンドを使用して、すべてのパイプライン実行をリスト表示してパイプラインが正常に実行されたことを確認します。

```
$ tkn pipelinerun list
```

出力には、パイプライン実行がリスト表示されます。

```
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. アプリケーションルートを取得します。

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

上記のコマンドの出力に留意してください。このルートを使用してアプリケーションにアクセスできます。

7. 直前のパイプラインのパイプラインリソースおよびサービスアカウントを使用して最後のパイプライン実行を再実行するには、以下を実行します。

```
$ tkn pipeline start build-and-deploy --last
```

関連情報

- git シークレットを使用したパイプラインの認証

1.7. トリガーのパイプラインへの追加

トリガーは、パイプラインがプッシュイベントやプル要求などの外部の GitHub イベントに応答できるようにします。アプリケーションのパイプラインをアセンブルし、起動した後に、**TriggerBinding**、**TriggerTemplate**、**Trigger**、および **EventListener** リソースを追加して GitHub イベントを取得します。

手順

1. 以下のサンプル **TriggerBinding** YAML ファイルの内容をコピーし、これを保存します。

```
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. **TriggerBinding** リソースを作成します。

```
$ oc create -f <triggerbinding-yaml-file-name.yaml>
```

または、**TriggerBinding** リソースを **pipelines-tutorial** Git リポジトリーから直接作成できます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/03_triggers/01_binding.yaml
```

3. 以下のサンプル **TriggerTemplate** YAML ファイルの内容をコピーし、これを保存します。

```
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-1.14
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1
    kind: PipelineRun
    metadata:
      generateName: build-deploy-$(tt.params.git-repo-name)-
    spec:
      taskRunTemplate:
        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
```

テンプレートは、ワークスペースのストレージボリュームを定義するための永続ボリューム要求 (PVC) を作成するためのボリューム要求テンプレートを指定します。そのため、データストレージを提供するために永続ボリューム要求 (PVC) を作成する必要はありません。

4. **TriggerTemplate** リソースを作成します。

```
$ oc create -f <triggertemplate-yaml-file-name.yaml>
```

または、**TriggerTemplate** リソースを **pipelines-tutorial** Git リポジトリーから直接作成できます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/03_triggers/02_template.yaml
```

5. 以下のサンプルの **Trigger** YAML ファイルの内容をコピーし、保存します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: vote-trigger
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
```

6. **Trigger** リソースを作成します。

```
$ oc create -f <trigger-yaml-file-name.yaml>
```

または、**Trigger** リソースを **pipelines-tutorial** Git リポジトリーから直接作成できます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/03_triggers/03_trigger.yaml
```

7. 以下のサンプル **EventListener** YAML ファイルの内容をコピーし、これを保存します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  triggers:
    - triggerRef: vote-trigger
```

または、トリガーカスタムリソースを定義していない場合は、トリガーの名前を参照する代わりに、バインディングおよびテンプレート仕様を **EventListener** YAML ファイルに追加します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
    template:
      ref: vote-app
```

8. 以下のコマンドを実行して **EventListener** リソースを作成します。

- セキュアな HTTPS 接続を使用して **EventListener** リソースを作成するには、以下を実行します。

a. ラベルを追加して、**Eventlistener** リソースへのセキュアな HTTPS 接続を有効にします。

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

b. **EventListener** リソースを作成します。

```
$ oc create -f <eventlistener-yaml-file-name.yaml>
```

または、**EvenListener** リソースを **pipelines-tutorial** Git リポジトリーから直接作成できます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.14/03_triggers/04_event_listener.yaml
```

c. re-encrypt TLS 終端でルートを作成します。

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

または、re-encrypt TLS 終端 YAML ファイルを作成して、セキュアなルートを作成できます。

セキュアなルートの re-encrypt TLS 終端 YAML の例

```
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) オブジェクトの名前で、63 文字に制限されます。

(3) **termination** フィールドは **reencrypt** に設定されます。これは、必要な唯一の **tls** フィールドです。

(4) 再暗号化に必要です。**destinationCACertificate** は CA 証明書を指定してエンドポイントの証明書を検証し、ルーターから宛先 Pod への接続のセキュリティーを保護します。サービスがサービス署名証明書を使用する場合または、管理者がデフォルトの CA 証明書をルーターに指定し、サービスにその CA により署名された証明書がある場合には、このフィールドは省略可能です。

他のオプションについては、**oc create route reencrypt --help** を参照してください。

- 非セキュアな HTTP 接続を使用して **EventListener** リソースを作成するには、以下を実行します。

  a. **EventListener** リソースを作成します。

  b. **EventListener** サービスを OpenShift Container Platform ルートとして公開し、これをアクセス可能にします。

```
$ oc expose svc el-vote-app
```

1.8. 複数の NAMESPACE を提供するようにイベントリスナーを設定する

注記

基本的な CI/CD パイプラインを作成する必要がある場合は、このセクションをスキップできます。ただし、デプロイメント戦略に複数の namespace が含まれる場合は、複数の namespace を提供するようにイベントリスナーを設定できます。

EvenListener オブジェクトの再利用性を高めるために、クラスター管理者は、複数の namespace にサービスを提供するマルチテナントイベントリスナーとして、これらのオブジェクトを設定およびデプロイできます。

手順

1. イベントリスナーのクラスター全体のフェッチ権限を設定します。

   a. **ClusterRoleBinding** オブジェクトおよび**EventListener** オブジェクトで使用するサービスアカウント名を設定します。たとえば、**el-sa**。

ServiceAccount.yaml の例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: el-sa
---
```

b. **ClusterRole.yaml** ファイルの **rules** セクションで、クラスター全体で機能するように、すべてのイベントリスナーデプロイメントに適切な権限を設定します。

ClusterRole.yaml の例

```
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. 適切なサービスアカウント名とクラスターロール名を使用して、クラスターロールバインディングを設定します。

ClusterRoleBinding.yaml の例

```
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. イベントリスナーの**spec**パラメーターに、サービスアカウント名 (**el-sa** など) を追加します。**namespaceSelector**パラメーターに、イベントリスナーがサービスを提供する namespace の名前を入力します。

EventListener.yaml の例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: namespace-selector-listener
spec:
  taskRunTemplate:
    serviceAccountName: el-sa
  namespaceSelector:
    matchNames:
    - default
    - foo
...
```

3. 必要な権限を持つサービスアカウントを作成します (例: **foo-trigger-sa**)。トリガーをロールバインドするために使用します。

   ServiceAccount.yaml の例

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo-trigger-sa
  namespace: foo
...
```

   RoleBinding.yaml の例

```
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. 適切なトリガーテンプレート、トリガーバインディング、およびサービスアカウント名を使用してトリガーを作成します。

   Trigger.yaml の例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  taskRunTemplate:
```

```
  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
...
```

1.9. WEBHOOK の作成

Webhook は、設定されたイベントがリポジトリーで発生するたびにイベントリスナーよって受信される HTTP POST メッセージです。その後、イベントペイロードはトリガーバインディングにマップされ、トリガーテンプレートによって処理されます。トリガーテンプレートは最終的に 1 つ以上のパイプライン実行を開始し、Kubernetes リソースの作成およびデプロイメントを実行します。

このセクションでは、フォークされた Git リポジトリー **pipelines-vote-ui** および **pipelines-vote-api** で Webhook URL を設定します。この URL は、一般に公開されている **EventListener** サービスルートを参照します。

注記

Webhook を追加するには、リポジトリーへの管理者権限が必要です。リポジトリーへの管理者アクセスがない場合は、Webhook を追加できるようにシステム管理者に問い合わせてください。

手順

1. Webhook URL を取得します。
   - セキュアな HTTPS 接続の場合:

     ```
     $ echo "URL: $(oc  get route el-vote-app --template='https://{{.spec.host}}')"
     ```

   - HTTP (非セキュアな) 接続の場合:

     ```
     $ echo "URL: $(oc  get route el-vote-app --template='http://{{.spec.host}}')"
     ```

     出力で取得した URL をメモします。
2. フロントエンドリポジトリーで Webhook を手動で設定します。
   a. フロントエンド Git リポジトリー **pipelines-vote-ui** をブラウザーで開きます。
   b. **Settings → Webhooks → Add Webhook** をクリックします。
   c. **Webhooks/Add Webhook** ページで以下を実行します。

i. 手順 1 の Webhook URL を **Payload URL** フィールドに入力します。

ii. **Content type** について **application/json** を選択します。

iii. シークレットを **Secret** フィールドに指定します。

iv. **Just the push event**が選択されていることを確認します。

v. **Active** を選択します。

vi. **Add Webhook** をクリックします。

3. バックエンドリポジトリー **pipelines-vote-api** について手順 2 を繰り返します。

1.10. パイプライン実行のトリガー

push イベントが Git リポジトリーで実行されるたびに、設定された Webhook はイベントペイロードを公開される **EventListener** サービスルートに送信します。アプリケーションの **EventListener** サービスはペイロードを処理し、これを関連する **TriggerBinding** および **TriggerTemplate** リソースのペアに渡します。**TriggerBinding** リソースはパラメーターを抽出し、**TriggerTemplate** リソースはこれらのパラメーターを使用して、リソースの作成方法を指定します。これにより、アプリケーションが再ビルドされ、再デプロイされる可能性があります。

このセクションでは、空のコミットをフロントエンドの **pipelines-vote-ui** リポジトリーにプッシュし、パイプライン実行をトリガーします。

手順

1. ターミナルから、フォークした Git リポジトリー **pipelines-vote-ui** のクローンを作成します。

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.14
```

2. 空のコミットをプッシュします。

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.14
```

3. パイプライン実行がトリガーされたかどうかを確認します。

```
$ tkn pipelinerun list
```

新規のパイプライン実行が開始されたことに注意してください。

1.11. ユーザー定義プロジェクトでの TRIGGERS のイベントリスナーのモニタリングの有効化

クラスター管理者は、イベントリスナーごとにサービスモニターを作成し、ユーザー定義のプロジェクトで **Triggers** サービスのイベントリスナーメトリクスを収集し、OpenShift Container Platform Web コンソールでそれらを表示することができます。HTTP リクエストを受信すると、**Triggers** サービスのイベントリスナーは 3 つのメトリクス (**eventlistener_http_duration_seconds**、**eventlistener_event_count**、および **eventlistener_triggered_resources**) を返します。

前提条件

- OpenShift Container Platform Web コンソールにログインしている。
- Red Hat OpenShift Pipelines Operator がインストールされている。
- ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. イベントリスナーごとに、サービスモニターを作成します。たとえば、**test** namespace の **github-listener** イベントリスナーのメトリクスを表示するには、以下のサービスモニターを作成します。

```
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. リクエストをイベントリスナーに送信して、サービスモニターをテストします。たとえば、空のコミットをプッシュします。

```
$ git commit -m "empty-commit" --allow-empty && git push origin main
```

3. OpenShift Container Platform Web コンソールで、**Administrator → Observe → Metrics** の順に移動します。
4. メトリクスを表示するには、名前で検索します。たとえば、**github-listener** イベントリスナーの **eventlistener_http_resources** メトリクスの詳細を表示するには、**eventlistener_http_resources** のキーワードを使用して検索します。

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

1.12. GITHUB INTERCEPTOR でのプルリクエスト機能の設定

GitHub Interceptor を使用すると、GitHub Webhook を検証およびフィルタリングするロジックを作成できます。たとえば、Webhook の発信元を検証し、指定された基準に基づいて着信イベントをフィルター処理できます。GitHub Interceptor を使用してイベントデータをフィルタリングする場合、Interceptor がフィールドで受け入れることができるイベントタイプを指定できます。Red Hat OpenShift Pipelines では、GitHub Interceptor の以下の機能を使用できます。

- 変更されたファイルに基づいてプルリクエストイベントをフィルタリングする
- 設定された GitHub 所有者に基づいてプルリクエストを検証する

1.12.1. GitHub Interceptor を使用したプルリクエストのフィルタリング

プッシュおよびプルイベント用に変更されたファイルに基づいて、GitHub イベントをフィルター処理できます。これは、Git リポジトリー内の関連する変更のみに対してパイプラインを実行するのに役立ちます。GitHub Interceptor は、変更されたすべてのファイルのコンマ区切りリストを追加し、CEL Interceptor を使用して、変更されたファイルに基づいて着信イベントをフィルタリングします。変更されたファイルのリストは、最上位の **extensions** フィールドのイベントペイロードの **changed_files** がプロパティーに追加されます。

前提条件

- Red Hat OpenShift Pipelines Operator がインストールされている。

手順

1. 以下のいずれかの手順を実行します。
   - パブリック GitHub リポジトリーの場合、以下に示す YAML 設定ファイルで **addChangedFiles** パラメーターの値を **true** に設定します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
          - name: "secretRef"
            value:
              secretName: github-secret
              secretKey: secretToken
          - name: "eventTypes"
            value: ["pull_request", "push"]
          - name: "addChangedFiles"
            value:
              enabled: true
        - ref:
```

```
          name: cel
        params:
        - name: filter
          value: extensions.changed_files.matches('controllers/')
...
```

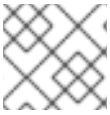
- プライベート GitHub リポジトリーの場合、**addChangedFiles** パラメーターの値を **true** に設定し、以下に示す YAML 設定ファイルでアクセストークンの詳細、**secretName**、および **secretKey** を指定します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
          - name: "secretRef"
            value:
              secretName: github-secret
              secretKey: secretToken
          - name: "eventTypes"
            value: ["pull_request", "push"]
          - name: "addChangedFiles"
            value:
              enabled: true
              personalAccessToken:
                secretName: github-pat
                secretKey: token
        - ref:
            name: cel
          params:
          - name: filter
            value: extensions.changed_files.matches('controllers/')
...
```

2. 設定ファイルを作成します。

1.12.2. GitHub Interceptors を使用したプルリクエストの検証

GitHub Interceptor を使用して、リポジトリー用に設定された GitHub 所有者に基づいてプルリクエストの処理を検証できます。この検証は、**PipelineRun** または **TaskRun** オブジェクトの不要な実行を防ぐのに役立ちます。GitHub Interceptor は、ユーザー名が所有者としてリストされている場合、または設定可能なコメントがリポジトリーの所有者によって発行された場合にのみ、プルリクエストを処理します。たとえば、所有者としてプルリクエストで /**ok-to-test** にコメントすると、**PipelineRun** または **TaskRun** がトリガーされます。



注記

所有者は、リポジトリーのルートにある **OWNERS** ファイルで設定されます。

前提条件

- Red Hat OpenShift Pipelines Operator がインストールされている。

手順

1. シークレットの文字列値を作成します。
2. その値で GitHub webhook を設定します。
3. シークレット値を含む **secretRef** という名前の Kubernetes シークレットを作成します。
4. Kubernetes シークレットを GitHub Interceptor への参照として渡します。
5. **owners** ファイルを作成し、承認者のリストを **approvers** セクションに追加します。
6. 以下のいずれかの手順を実行します。
   - パブリック GitHub リポジトリーの場合、以下に示す YAML 設定ファイルで **githubOwners** パラメーターの値を **true** に設定します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-owners-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
            - name: "secretRef"
              value:
                secretName: github-secret
                secretKey: secretToken
            - name: "eventTypes"
              value: ["pull_request", "issue_comment"]
            - name: "githubOwners"
              value:
                enabled: true
                checkType: none
...
```

   - プライベート GitHub リポジトリーの場合、**githubOwners** パラメーターの値を **true** に設定し、以下に示す YAML 設定ファイルでアクセストークンの詳細、**secretName**、および **secretKey** を指定します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
```

```
metadata:
  name: github-owners-listener
spec:
  triggers:
    - name: github-listener
      interceptors:
        - ref:
            name: "github"
            kind: ClusterInterceptor
            apiVersion: triggers.tekton.dev
          params:
            - name: "secretRef"
              value:
                secretName: github-secret
                secretKey: secretToken
            - name: "eventTypes"
              value: ["pull_request", "issue_comment"]
            - name: "githubOwners"
              value:
                enabled: true
                personalAccessToken:
                  secretName: github-token
                  secretKey: secretToken
                checkType: all
...
```

注記

checkType パラメーターは、認証が必要な GitHub 所有者を指定するために使用されます。その値を **orgMembers**、**repoMembers**、または **all** に設定できます。

7. 設定ファイルを作成します。

1.13. 関連情報

- Pipelines as Code をアプリケーションのソースコードとともに同じリポジトリーに含める方法は、 Pipelines as Code について を参照してください。
- **Developer** パースペクティブでのパイプラインの詳細は、Web コンソールでの OpenShift パイプラインの操作 セクションを参照してください。
- Security Context Constraints (SCC) の詳細は、セキュリティーコンテキスト制約の管理 セクションを参照してください。
- 再利用可能なタスクの追加の例については、OpenShift Catalog リポジトリーを参照してください。さらに、Tekton プロジェクトで Tekton Catalog を参照することもできます。
- 再利用可能なタスクとパイプライン用に Tekton Hub のカスタムインスタンスをインストールしてデプロイするには、Red Hat OpenShift Pipelines での Tekton Hub の使用 を参照してください。
- re-encrypt TLS 終端の詳細は、再暗号化終端 を参照してください。

- セキュリティー保護されたルートの詳細は、セキュリティー保護されたルート セクションを参照してください。

第2章 WEB コンソールでの RED HAT OPENSHIFT PIPELINES の使用

Administrator または **Developer** パースペクティブを使用して、OpenShift Container Platform Web コンソールの **Pipelines** ページから **Pipeline**、**PipelineRun**、**Repository** オブジェクトを作成および変更できます。Web コンソールの **Developer** パースペクティブの **+Add** ページを使用して、ソフトウェアデリバリープロセスの CI/CD パイプラインを作成することもできます。

2.1. DEVELOPER パースペクティブで RED HAT OPENSHIFT PIPELINES を使用する

Developer パースペクティブでは、**+Add** ページからパイプラインを作成するための以下のオプションにアクセスできます。

- **Add → Pipeline → Pipeline Builder** オプションを使用して、アプリケーションのカスタマイズされたパイプラインを作成します。
- **+Add → From Git** オプションを使用して、アプリケーション作成時にパイプラインテンプレートおよびリソースを使用してパイプラインを作成します。

アプリケーションのパイプラインの作成後に、**Pipelines** ビューでデプロイされたパイプラインを表示し、これらと視覚的に対話できます。**Topology** ビューを使用して、**From Git** オプションを使用して作成されたパイプラインと対話することもできます。**パイプライン** ビルダーを使用して作成されたパイプラインを **トポロジー** ビューで表示するには、カスタムラベルを適用する必要があります。

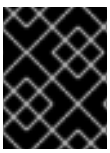
前提条件

- OpenShift Container Platform クラスターにアクセスでき、**開発者** パースペクティブ に切り替えている。
- クラスターに OpenShift Pipelines Operator がインストールされている。
- クラスター管理者か、create および edit パーミッションを持つユーザーである。
- プロジェクトを作成している。

2.1.1. Pipeline Builder を使用した Pipeline の構築

コンソールの **Developer** パースペクティブで、**+Add → Pipeline → Pipeline Builder** オプションを使用して以下を実行できます。

- **Pipeline ビルダー** または **YAML ビュー** のいずれかを使用してパイプラインを設定します。
- 既存のタスクおよびクラスタータスクを使用して、パイプラインフローを構築します。OpenShift Pipelines Operator をインストールする際に、再利用可能なパイプラインクラスタータスクをクラスターに追加します。

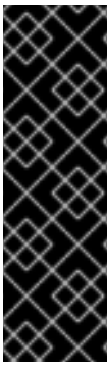


重要

Red Hat OpenShift Pipelines 1.10 では、クラスタータスク機能は非推奨であり、将来のリリースで削除される予定です。

- パイプライン実行に必要なリソースタイプを指定し、必要な場合は追加のパラメーターをパイプラインに追加します。

- パイプラインの各タスクのこれらのパイプラインリソースを入力および出力リソースとして参照します。
- 必要な場合は、タスクのパイプラインに追加されるパラメーターを参照します。タスクのパラメーターは、Task の仕様に基づいて事前に設定されます。
- Operator によってインストールされた、再利用可能なスニペットおよびサンプルを使用して、詳細なパイプラインを作成します。
- 設定済みのローカル Tekton Hub インスタンスからタスクを検索して追加します。



重要

開発者の観点では、キュレートされた独自のタスクセットを使用して、カスタマイズされたパイプラインを作成できます。タスクを開発者コンソールから直接検索、インストール、およびアップグレードするには、クラスター管理者がローカルの Tekton Hub インスタンスをインストールしてデプロイし、そのハブを OpenShift Container Platform クラスターにリンクする必要があります。詳細は、**追加のリソース** セクションの **OpenShift Pipeline での Tekton Hub の使用**セクションを参照してください。ローカル Tekton Hub インスタンスをデプロイしない場合、デフォルトでは、クラスタータスク、ネームスペースタスク、およびパブリック Tekton Hub タスクにのみアクセスできます。

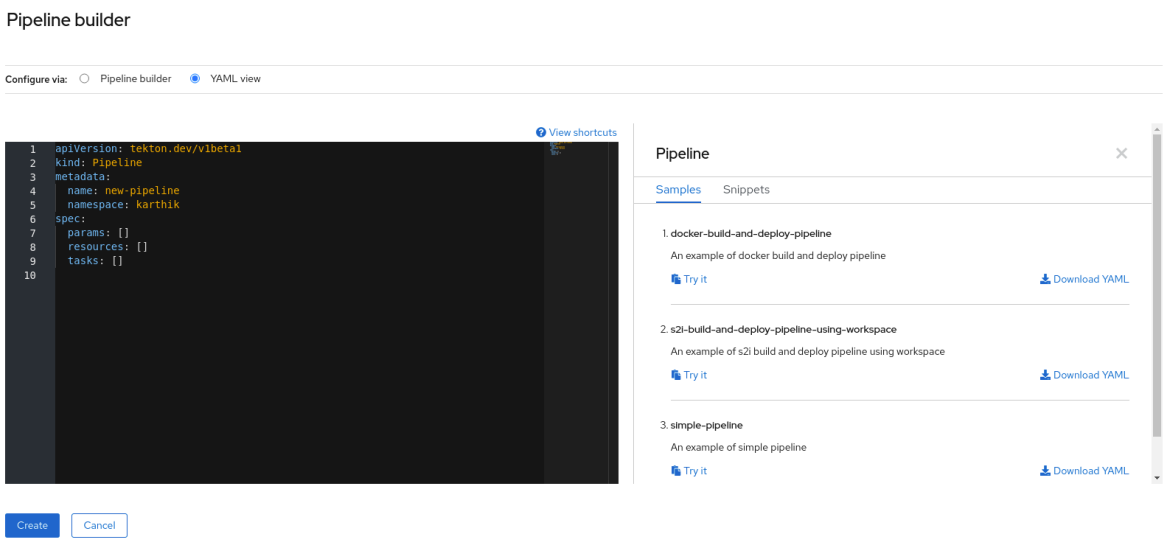
手順

1. **Developer** パースペクティブの **+Add** ビューで、**Pipeline** タイルをクリックし、**Pipeline Builder** ページを表示します。
2. **Pipeline ビルダー** ビューまたは **YAML ビュー** のいずれかを使用して、パイプラインを設定します。

注記

Pipeline ビルダー ビューは、限られた数のフィールドをサポートしますが、**YAML ビュー** は利用可能なすべてのフィールドをサポートします。オプションで、Operator によってインストールされた、再利用可能なスニペットおよびサンプルを使用して、詳細な Pipeline を作成することもできます。

図2.1 YAML ビュー



3. **Pipeline builder** を使用してパイプラインを設定します。

   a. **Name** フィールドにパイプラインの一意の名前を入力します。

   b. **Tasks** セクションで、以下を実行します。

      i. **Add task** をクリックします。

      ii. クイック検索フィールドを使用してタスクを検索し、表示されたリストから必要なタスクを選択します。

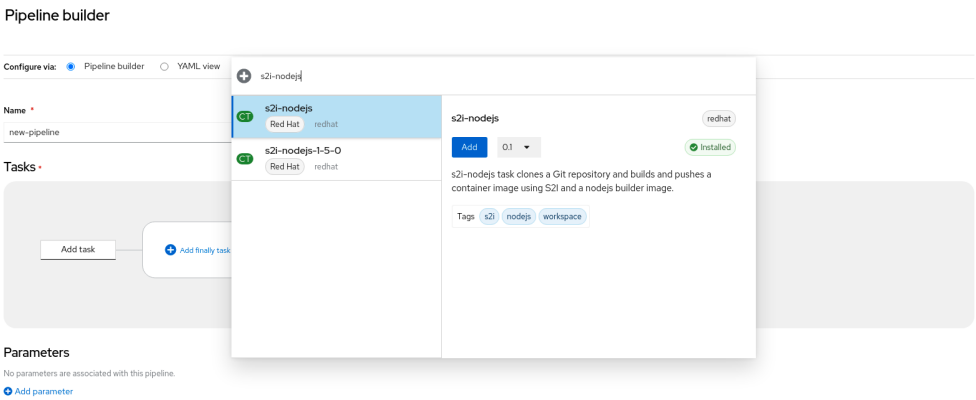
      iii. **Add** または **Install and add** をクリックします。この例では、**s2i-nodejs** タスクを使用します。

注記

検索のリストには、Tekton Hub タスクおよび、クラスターで利用可能なタスクがすべて含まれます。また、タスクがすでにインストールされている場合は、タスク追加用の **Add** が表示され、それ以外の場合は、タスクのインストールおよび追加用の **Install and add** が表示されます。更新されたバージョンで同じタスクを追加する場合は、**Update and add** が表示されます。

- 連続するタスクをパイプラインに追加するには、以下を実行します。
  - タスクの右側にあるプラスアイコンをクリックし、**Add task** をクリックします。
  - クイック検索フィールドを使用してタスクを検索し、表示されたリストから必要なタスクを選択します。
  - **Add** または **Install and add** をクリックします。

図2.2 Pipeline Builder



- 最終タスクを追加するには、以下を実行します。
  - **Add finally task** → **Add task** の順にクリックします。
  - クイック検索フィールドを使用してタスクを検索し、表示されたリストから必要なタスクを選択します。
  - **Add** または **Install and add** をクリックします。

c. **Resources** セクションで、**Add Resources** をクリックし、パイプライン実行用のリソースの名前およびタイプを指定します。これらのリソースは、パイプラインのタスクによって入力および出力として使用されます。この例では、以下のようになります。

i. 入力リソースを追加します。**Name** フィールドに **Source** を入力してから、**Resource Type** ドロップダウンリストから **Git** を選択します。

ii. 出力リソースを追加します。**Name** フィールドに **Img** を入力してから、**Resource Type** ドロップダウンリストから **イメージ** を選択します。

注記

リソースが見つからない場合には、タスクの横に赤のアイコンが表示されます。

d. オプション: タスクの **Parameters** は、タスクの仕様に基づいて事前に設定されます。必要な場合は、**Parameters** セクションの **Add Parameters** リンクを使用して、パラメーターを追加します。

e. **Workspaces** セクションで、**Add workspace** をクリックし、**Name** フィールドに一意のワークスペース名を入力します。複数のワークスペースをパイプラインに追加できます。

f. **Tasks** セクションで、**s2i-nodejs** タスクをクリックし、タスクの詳細情報が含まれるサイドパネルを表示します。タスクのサイドパネルで、**s2i-nodejs** タスクのリソースおよびパラメーターを指定します。

i. 必要な場合は、**Parameters** セクションで、$(params.<param-name>) 構文を使用して、デフォルトのパラメーターにパラメーターをさらに追加します。

ii. **Image** セクションで、**Resources** セクションで指定されているように **Img** を入力します。

iii. **Workspace** セクションの **source** ドロップダウンからワークスペースを選択します。

g. リソース、パラメーター、およびワークスペースを **openshift-client** タスクに追加します。

4. **Create** をクリックし、**Pipeline Details** ページでパイプラインを作成し、表示します。

5. **Actions** ドロップダウンメニューをクリックしてから **Start** をクリックし、**Start Pipeline** ページを表示します。

6. **Workspace** セクションは、以前に作成したワークスペースをリスト表示します。それぞれのドロップダウンを使用して、ワークスペースのボリュームソースを指定します。**Empty Directory**、**Config Map**、**Secret**、**PersistentVolumeClaim**、または **VolumeClaimTemplate** のオプションを使用できます。

2.1.2. アプリケーションと共に OpenShift Pipelines を作成する

アプリケーションと共にパイプラインを作成するには、**Developer** パースペクティブの **Add+** ビューで、**From Git** オプションを使用します。使用可能なすべてのパイプラインを表示し、コードのインポートまたはイメージのデプロイ中に、アプリケーションの作成に使用するパイプラインを選択できます。

Tekton Hub 統合はデフォルトで有効になっており、クラスターでサポートされている Tekton Hub からのタスクを確認できます。管理者は Tekton Hub 統合をオプトアウトでき、Tekton Hub タスクは表示さ

れなくなります。生成されたパイプラインに Webhook URL が存在するかどうかを確認することもできます。**+Add**フローを使用して作成されたパイプラインにデフォルトの Webhook が追加され、Topology ビューで選択したリソースのサイドパネルに URL が表示されます。

詳細は、Developer パースペクティブを使用したアプリケーションの作成 を参照してください。

2.1.3. パイプラインを含む GitHub リポジトリーの追加

Developer パースペクティブでは、パイプラインを含む GitHub リポジトリーを OpenShift Container Platform クラスターに追加できます。これにより、プッシュリクエストやプルリクエストなどの関連する Git イベントがトリガーされたときに、クラスター上の GitHub リポジトリーからパイプラインとタスクを実行できます。

注記

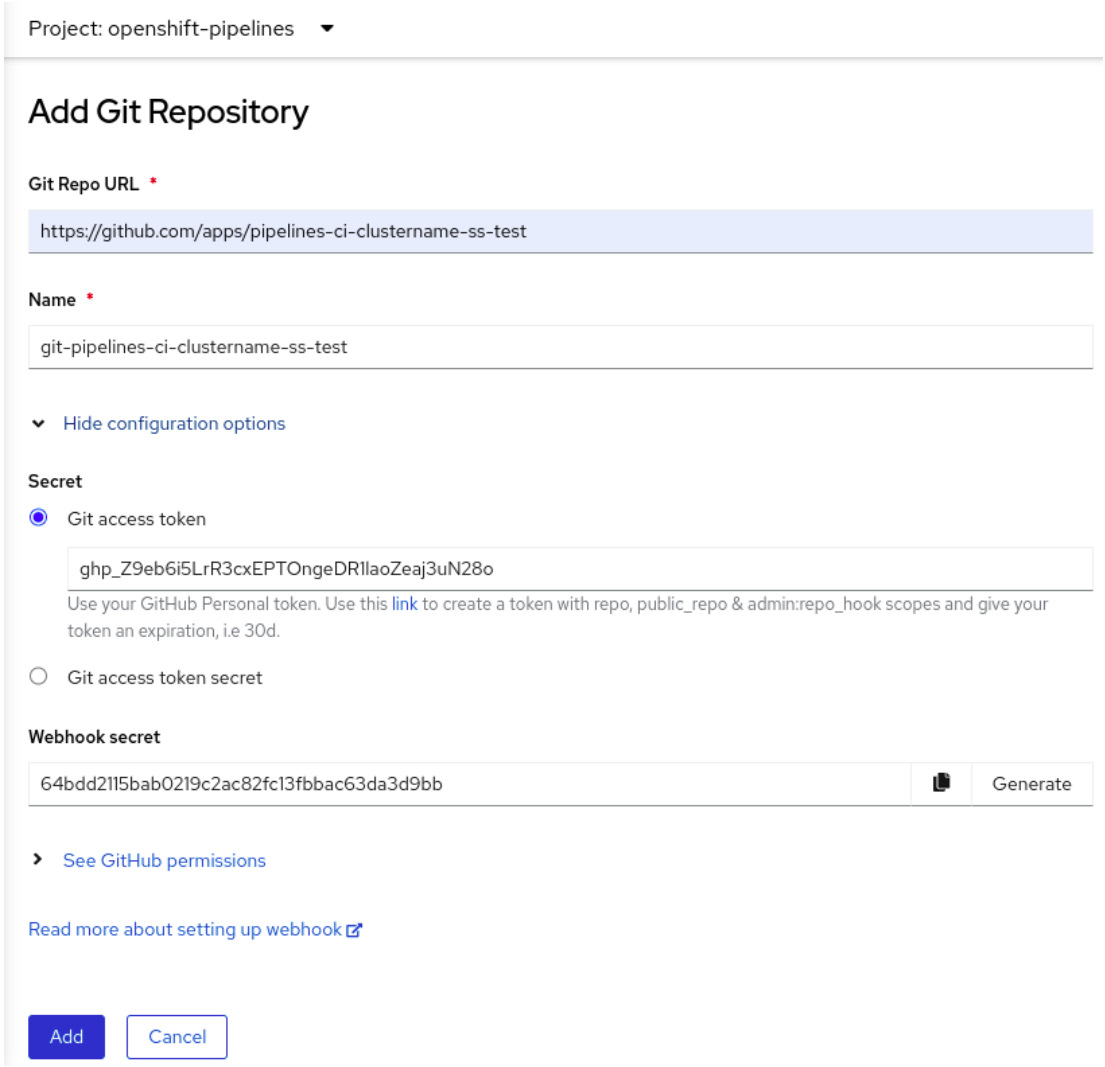
パブリックおよびプライベートの GitHub リポジトリーの両方を追加できます。

前提条件

- クラスター管理者が必要な GitHub アプリケーションを管理者パースペクティブで設定していることを確認してください。

手順

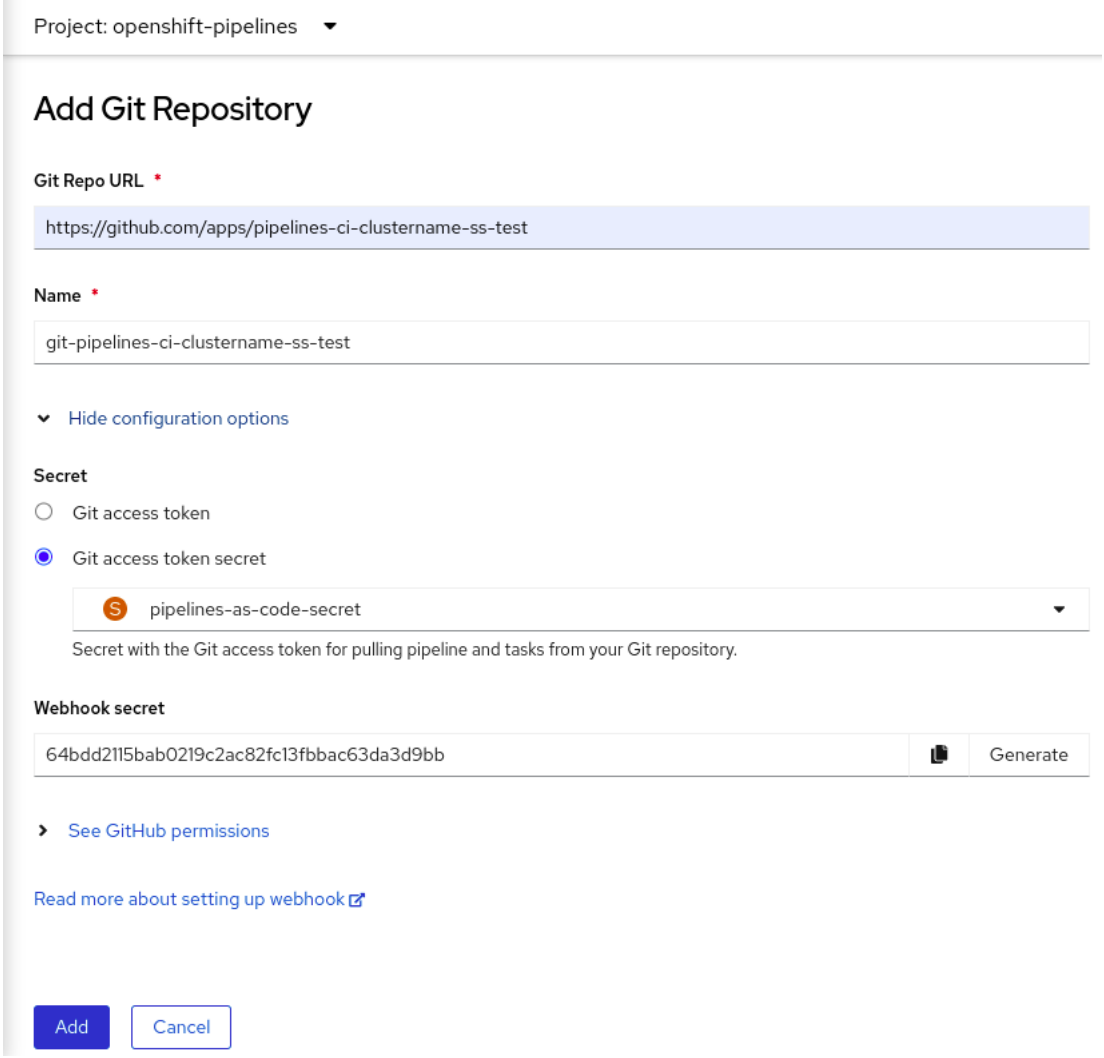
1. **Developer** パースペクティブで、GitHub リポジトリーを追加する namespace またはプロジェクトを選択します。
2. 左側のナビゲーションペインを使用して **パイプライン** に移動します。
3. **Pipelines** ページの右側にある **Create → Repository** をクリックします。
4. **Git Repo URL** を入力すると、コンソールが自動的にリポジトリー名を取得します。
5. **設定オプションを表示を** クリックします。デフォルトでは、**Setup a webhook** というオプションが 1 つだけ表示されます。GitHub アプリケーションが設定されている場合は、次の 2 つのオプションが表示されます。
   - **Use GitHub App**: リポジトリーに GitHub アプリケーションをインストールするには、このオプションを選択します。
   - **Setup a webhook**: Webhook を GitHub アプリケーションに追加するには、このオプションを選択します。
6. **Secret** セクションで次のいずれかのオプションを使用して Webhook を設定します。
   - **Git アクセストークン** を使用して Webhook をセットアップします。
     a. 個人用アクセストークンを入力します。
     b. **Webhook シークレット** フィールドに対応する **生成** をクリックして、新しい Webhook シークレットを生成します。



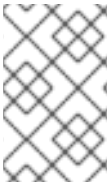
注記

個人用アクセストークンを持っておらず、新しいトークンを作成する場合は、**Git access token** フィールドの下のリンクをクリックできます。

- **Git access token secret** を使用して Webhook をセットアップします。
  - ドロップダウンリストから namespace のシークレットを選択します。選択したシークレットに応じて、Webhook シークレットが自動的に生成されます。



7. Webhook シークレットの詳細を GitHub リポジトリーに追加します。
    a. **Webhook URL** をコピーし、GitHub リポジトリー設定に移動します。
    b. **Webhooks → Add webhook** をクリックします。
    c. 開発者コンソールから **Webhook URL** をコピーし、GitHub リポジトリー設定の **Payload URL** フィールドに貼り付けます。
    d. **Content type** を選択します。
    e. 開発者コンソールから **Webhook secret** をコピーし、GitHub リポジトリー設定の **Secret** フィールドに貼り付けます。
    f. **SSL 検証** オプションのいずれかを選択します。
    g. この Webhook をトリガーするイベントを選択します。
    h. **Add webhook** をクリックします。
8. 開発者コンソールに戻り、**Add** をクリックします。
9. 実行する手順の詳細を確認し、**Close** をクリックします。
10. 作成したリポジトリーの詳細を表示します。



注記

Import from Git を使用してアプリケーションをインポートし、Git リポジトリーに **.tekton** ディレクトリーがある場合、アプリケーションの **pipelines-as-code** を設定できます。

2.1.4. 開発者パースペクティブを使用したパイプラインの使用

Developer パースペクティブの **Pipelines** ビューは、以下の詳細と共にプロジェクトのすべてのパイプラインをリスト表示します。

- パイプラインが作成された namespace
- 最後のパイプライン実行
- パイプライン実行のタスクのステータス
- パイプライン実行のステータス
- 最後のパイプライン実行の作成時間

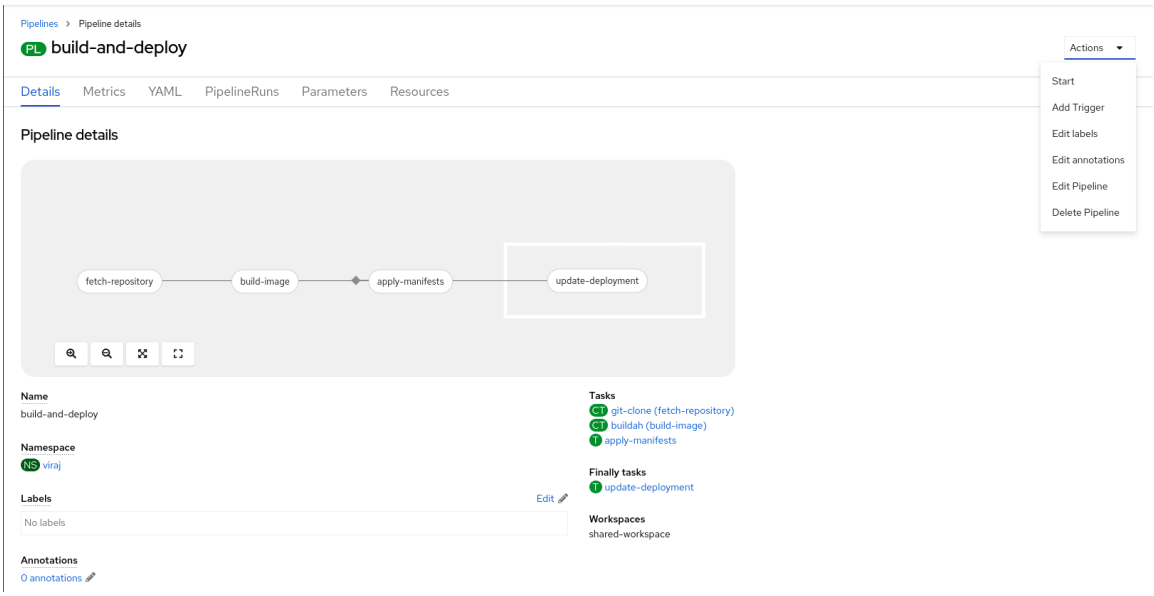
手順

1. **Developer** パースペクティブの **Pipelines** ビューで、**Project** ドロップダウンリストからプロジェクトを選択し、そのプロジェクトのパイプラインを表示します。
2. 必要なパイプラインをクリックし、**Pipeline Details** ページを表示します。
   デフォルトでは、**Details** タブには、すべての **serial** タスク、**parallel** タスク、**finally** タスク、およびパイプライン **when** の式がすべて視覚的に表示されます。タスクと **finally** タスクは、ページの右下に一覧表示されます。

   タスクの詳細を表示するには、一覧表示されている **Tasks** および **Finally** タスクをクリックします。さらに、以下を実行できます。

   - **パイプライン詳細** の視覚化の左下隅に表示される標準アイコンを使用して、ズームイン、ズームアウト、画面サイズの自動調整、およびビューのリセット機能を使用します。
   - マウスホイールを使用して、パイプラインビジュアライゼーションのズーム係数を変更します。
   - タスクにカーソルを合わせ、タスクの詳細を表示します。

図2.3 Pipeline の詳細



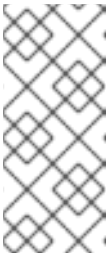
3. オプション: **Pipeline details** ページで、 **Metrics** タブをクリックして、パイプラインに関する以下の情報を表示します。

   - **Pipeline 成功比率**
   - **Pipeline Run の数**
   - **Pipeline Run の期間**
   - **Task Run Balancing**
     この情報を使用して、パイプラインのワークフローを改善し、パイプラインのライフサイクルの初期段階で問題をなくすことができます。

4. オプション: **YAML** タブをクリックし、パイプラインの YAML ファイルを編集します。

5. オプション: **Pipeline Runs** タブをクリックして、パイプラインの完了済み、実行中、または失敗した実行を確認します。
   Pipeline Runs タブでは、パイプライン実行、タスクのステータス、および失敗したパイプライン実行のデバッグ用のリンクの詳細が表示されます。Options メニュー を使用して、実行中のパイプラインを停止するか、以前のパイプライン実行と同じパラメーターとリソースを使用してパイプラインを再実行するか、パイプライン実行を削除します。

   - 必要なパイプラインをクリックし、**Pipeline Run details** ページを表示します。デフォルトでは、**Details** タブには、すべてのシリアルタスク、並列タスク、**finally** タスク、およびパイプライン実行の式がすべて視覚的に表示されます。実行に成功すると、ページ下部の **Pipeline Run results** ペインに表示されます。さらに、クラスターでサポートされている Tekton Hub からのタスクのみを表示できます。タスクを見ながら、その横にあるリンクをクリックして、タスクのドキュメントにジャンプできます。



注記

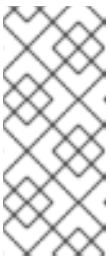
Pipeline Run Details ページの **Details** セクションには、失敗したパイプライン実行の **Log Snippet** (ログスニペット) が表示されます。**Log Snippet** (ログスニペット) は、一般的なエラーメッセージとログのスニペットを提供します。**Logs** セクションへのリンクでは、失敗した実行に関する詳細へのクイックアクセスを提供します。

- **Pipeline Run details**ページで、**Task Runs**タブをクリックして、タスクの完了、実行、および失敗した実行を確認します。
  Task Runsタブは、タスク実行に関する情報と、そのタスクおよび Pod へのリンクと、タスク実行のステータスおよび期間を提供します。Options メニュー を使用してタスク実行を削除します。

注記

TaskRuns リストページには **Manage columns** ボタンがあり、これを使用して **Duration** 列を追加することもできます。

- 必要なタスク実行をクリックして、**Task Run details**ページを表示します。実行に成功すると、ページ下部の **Task Run results**ペインに表示されます。



注記

Task Run detailsページの **Details** セクションには、失敗したパイプライン実行の **Log Snippet**(ログスニペット) が表示されます。**Log Snippet**(ログスニペット) は、一般的なエラーメッセージとログのスニペットを提供します。**Logs** セクションへのリンクでは、失敗した実行に関する詳細へのクイックアクセスを提供します。

6. **Parameters** タブをクリックして、パイプラインに定義されるパラメーターを表示します。必要に応じて追加のパラメーターを追加するか、編集することもできます。

7. **Resources** タブをクリックして、パイプラインで定義されたリソースを表示します。必要に応じて関連情報を追加するか、編集することもできます。

2.1.5. Pipelines ビューからパイプラインを開始する

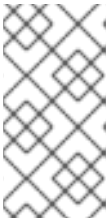
パイプラインの作成後に、これを開始し、これに含まれるタスクを定義されたシーケンスで実行できるようにする必要があります。パイプラインを **Pipelines** ビュー、**Pipeline Details** ページ、または **Topology** ビューから開始できます。

手順

Pipelines ビューを使用してパイプラインを開始するには、以下を実行します。

1. **Developer** パースペクティブの **Pipelines** ビューで、パイプラインに隣接する **Options** メニューで、**Start** を選択します。

2. **Start Pipeline** ダイアログボックスは、パイプライン定義に基づいて **Git Resources** および **Image Resources** を表示します。



注記

From Git オプションを使用して作成されるパイプラインの場合、**Start Pipeline** ダイアログボックスでは **Parameters** セクションに **APP_NAME** フィールドも表示され、ダイアログボックスのすべてのフィールドがパイプラインテンプレートによって事前に入力されます。

a. namespace にリソースがある場合、**Git Resources** および **Image Resources** フィールドがそれらのリソースで事前に設定されます。必要な場合は、ドロップダウンを使用して必要なリソースを選択または作成し、Pipeline Run インスタンスをカスタマイズします。

3. オプション: **Advanced Options** を変更し、認証情報を追加して、指定されたプライベート Git サーバーまたはイメージレジストリーを認証します。

   a. **Advanced Options** で **Show Credentials Options** をクリックし、**Add Secret** を選択します。

   b. **Create Source Secret** セクションで、以下を指定します。

      i. シークレットの一意の **シークレット名**。

      ii. **Designated provider to be authenticated** セクションで、**Access to** フィールドで認証されるプロバイダー、およびベース **Server URL** を指定します。

      iii. **Authentication Type** を選択し、認証情報を指定します。

      - Authentication Type **Image Registry Credentials** については、認証する **Registry Server Address** を指定し、**Username**、**Password**、および **Email** フィールドに認証情報を指定します。
        追加の **Registry Server Address** を指定する必要がある場合は、**Add Credentials** を選択します。

      - Authentication Type **Basic Authentication** については、**UserName** および **Password or Token** フィールドの値を指定します。

      - Authentication Type **SSH Keys** については、**SSH Private Key** フィールドの値を指定します。

      注記

      Basic 認証および SSH 認証には、以下のようなアノテーションを使用できます。

      - **tekton.dev/git-0: https://github.com**
      - **tekton.dev/git-1: https://gitlab.com**.

      iv. シークレットを追加するためにチェックマークを選択します。

   パイプラインのリソースの数に基づいて、複数のシークレットを追加できます。

4. **Start** をクリックしてパイプラインを開始します。

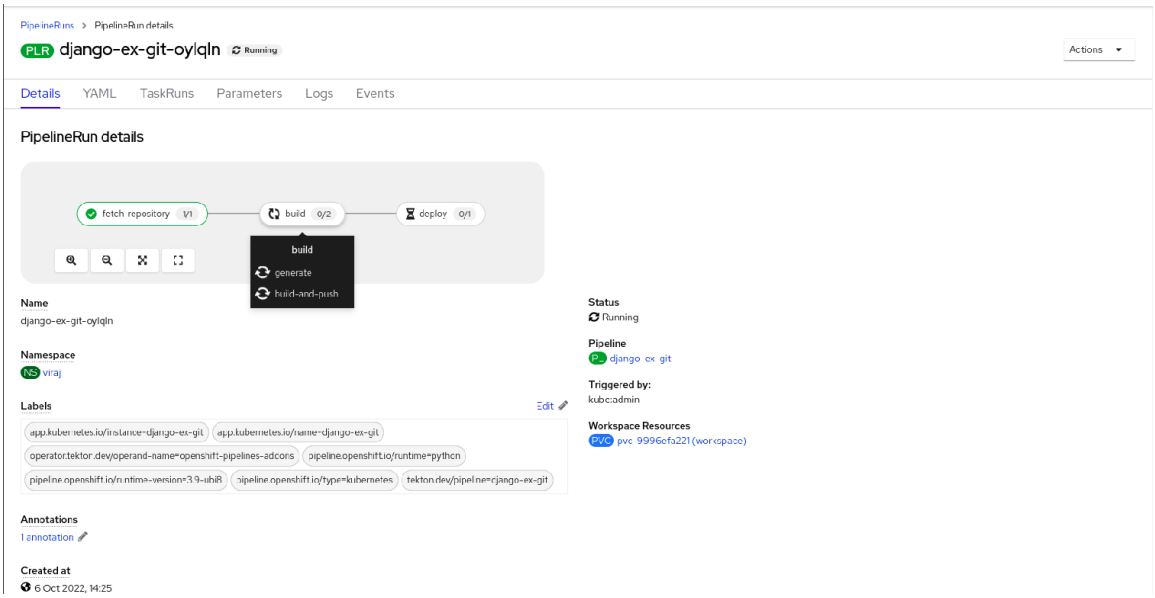
5. **PipelineRun details** ページには、実行されるパイプラインが表示されます。パイプラインが開始すると、タスクおよび各タスク内のステップが実行されます。以下を行うことができます。

   - **PipelineRun 詳細** ページビジュアライゼーションの左下隅にある標準アイコンを使用して、ズームイン、ズームアウト、画面サイズの自動調整、およびビューのリセット機能を使用します。

   - マウスホイールを使用して、パイプライン実行の視覚化のズーム係数を変更します。特定のズーム要素では、タスクの背景色が変更され、エラーまたは警告のステータスが示されます。

- タスクにカーソルを合わせると、各ステップの実行にかかった時間、タスク名、タスクステータスなどの詳細が表示されます。
- タスクバッジにカーソルを合わせ、完了したタスクとタスクの合計数を確認します。
- タスクをクリックし、タスクの各ステップのログを表示します。
- **Logs** タブをクリックして、タスクの実行シーケンスに関連するログを表示します。該当するボタンを使用して、ペインをデプロイメントし、ログを個別に、または一括してダウンロードすることもできます。
- **Events** タブをクリックして、パイプライン実行で生成されるイベントのストリームを表示します。
  Task Runs、**Logs**、および **Events** タブを使用すると、失敗したパイプラインの実行またはタスクの実行のデバッグに役立ちます。

図2.4 パイプライン実行の詳細



2.1.6. Topology ビューからパイプラインを開始する

From Git オプションを使用して作成されるパイプラインの場合、**Topology** ビューを使用して、開始後のパイプラインと対話することができます。

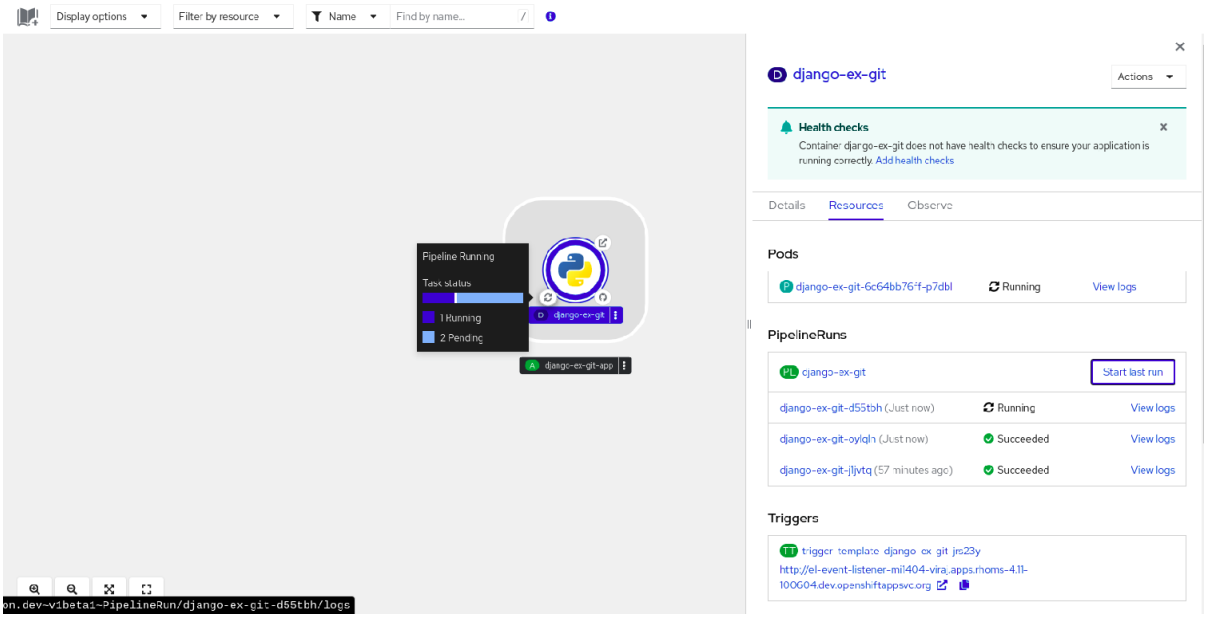
注記

Topology ビューで **Pipeline Builder** を使用して作成されるパイプラインを表示するには、パイプラインのラベルをカスタマイズし、パイプラインをアプリケーションのワークロードにリンクします。

手順

1. 左側のナビゲーションパネルで **Topology** をクリックします。
2. アプリケーションをクリックして、サイドパネルに **Pipeline Runs** を表示します。
3. **Pipeline Runs** で、**Start Last Run** をクリックして、前のパイプラインと同じパラメーターとリソースを使用して新しいパイプラインの実行を開始します。このオプションは、パイプライン実行が開始されていない場合は無効になります。パイプラインの作成時にパイプラインの実行を開始することもできます。

図2.5 Topology ビューのパイプライン



Topology ページで、アプリケーションの左側にカーソルを合わせると、パイプライン実行のステータスが表示されます。パイプラインが追加された後、左下のアイコンは、関連付けられたパイプラインがあることを示します。

2.1.7. Topology ビューからのパイプラインとの対話

Topology ページのアプリケーションノードのサイドパネルには、パイプライン実行のステータスが表示され、対話することができます。

- パイプラインの実行が自動的に開始されない場合、サイドパネルにパイプラインを自動的に開始できないというメッセージが表示されるため、手動で開始する必要があります。
- パイプラインが作成されたが、ユーザーがパイプラインを開始していない場合、そのステータスは Not started になります。ユーザーが、**Not started** ステータスアイコンをクリックすると、**Topology** ビューに start ダイアログボックスが開きます。
- パイプラインにビルドまたはビルド設定がない場合、**Builds**セクションは表示されません。パイプラインとビルド設定がある場合は、**Builds セクション** が表示されます。
- 特定のタスク実行でパイプライン実行が失敗すると、サイドパネルに **Log Snippet** が表示されます。**Resources** タブの **Pipeline Runs** セクションに **Log Snippet** を表示できます。これは、一般的なエラーメッセージとログのスニペットを提供します。**Logs** セクションへのリンクでは、失敗した実行に関する詳細へのクイックアクセスを提供します。

2.1.8. Pipeline の編集

Web コンソールの **Developer** パースペクティブを使用して、クラスター内のパイプラインを編集できます。

手順

1. **Developer** パースペクティブの **Pipelines** ビューで、編集する必要のある Pipeline を選択し、Pipeline の詳細を表示します。**Pipeline Details** ページで **Actions** をクリックし、**Edit Pipeline** を選択します。
2. **パイプラインビルダー** ページでは、次のタスクを実行できます。

- 追加のタスク、パラメーター、またはリソースをパイプラインに追加します。
- 変更するタスクをクリックして、サイドパネルにタスクの詳細を表示し、表示名、パラメーター、リソースなどの必要なタスクの詳細を変更します。
- または、Task を削除するには、Task をクリックし、サイドパネルで **Actions** をクリックし、**Remove Task** を選択します。

3. **Save** をクリックして変更された Pipeline を保存します。

2.1.9. Pipeline の削除

Web コンソールの **Developer** パースペクティブを使用して、クラスターの Pipeline を削除できます。

手順

1. **Developer** パースペクティブの **Pipelines** ビューで、Pipeline に隣接する **Options** メニューをクリックし、**Delete Pipeline** を選択します。
2. **Delete Pipeline** 確認プロンプトで、**Delete** をクリックし、削除を確認します。

2.2. 関連情報

- OpenShift Pipelines での Tekton Hub の使用

2.3. ADMINISTRATOR パースペクティブでのパイプラインテンプレートの作成

クラスター管理者は、開発者がクラスターでパイプラインを作成するときに再利用できるパイプラインテンプレートを作成できます。

前提条件

- クラスター管理者権限で OpenShift Container Platform クラスターにアクセスでき、**Administrator** パースペクティブに切り替えている。
- OpenShift Pipelines Operator がクラスターにインストールされている。

手順

1. **Pipelines** ページに移動し、既存のパイプラインテンプレートを表示します。
2. アイコンをクリックして **Import YAML** ページに移動します。
3. パイプラインテンプレートの YAML を追加します。テンプレートには、以下の情報が含まれている必要があります。

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
# ...
```

```
  namespace: openshift (1)
  labels:
    pipeline.openshift.io/runtime: <runtime> (2)
    pipeline.openshift.io/type: <pipeline-type> (3)
# ...
```

(1) テンプレートは **openshift** namespace に作成する必要があります。

(2) テンプレートには **pipeline.openshift.io/runtime** ラベルが含まれている必要があります。このラベルで許可されるランタイム値は、**nodejs**、**golang**、**dotnet**、**java**、**php**、**ruby**、**perl**、**python**、**nginx**、および **httpd** です。

(3) テンプレートには、**pipeline.openshift.io/type:** ラベルが含まれている必要があります。このラベルで許可されるタイプ値は、**openshift**、**knative**、および **kubernetes** です。

4. **Create** をクリックします。パイプラインを作成すると、**Pipeline details** ページが表示されます。ここでは、Pipeline 情報の表示や編集が可能です。

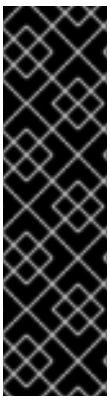
2.4. WEB コンソールのパイプライン実行に関する統計情報

Web コンソールでパイプラインの実行に関連する統計を表示できます。

統計情報を表示するには、次の手順を完了する必要があります。

- Tekton Results をインストールします。Tekton Results のインストールの詳細は、**関連情報** セクションの **OpenShift Pipelines の可観測性のための Tekton Results の使用** を参照してください。
- OpenShift Pipelines コンソールプラグインを有効にします。

統計情報は、すべてのパイプラインをまとめて、または個別のパイプラインごとに利用できます。



重要

OpenShift Pipelines Pipelines コンソールプラグインはテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

関連情報

- OpenShift Pipelines の可観測性のために Tekton 結果を使用する

2.4.1. OpenShift Pipelines コンソールプラグインの有効化

統計情報を表示するには、まず OpenShift Pipelines コンソールプラグインを有効にする必要があります。

前提条件

- Red Hat OpenShift Pipelines Operator がクラスターにインストールされている。
- クラスター管理者のパーミッションで Web コンソールにログインしている。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** を選択します。
2. Operator の表で **Red Hat OpenShift Pipelines** をクリックします。
3. 画面の右側のペインで、**Console plugin** の下のステータスラベルを確認します。ラベルは **Enabled** または **Disabled** のいずれかになります。
4. ラベルが **Disabled** の場合は、このラベルをクリックします。表示されるウィンドウで、**Enable** を選択し、**Save** をクリックします。

2.4.2. すべてのパイプラインの統計をまとめて表示

システム上のすべてのパイプラインに関連する統合統計情報を表示できます。

前提条件

- Red Hat OpenShift Pipelines Operator がクラスターにインストールされている。
- Tekton Results をインストールしている。
- OpenShift Pipelines Web コンソールプラグインがインストールされている。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Pipelines → Overview** を選択します。
   統計の概要が表示されます。この概要には、**一定期間におけるパイプライン実行の数とステータスを反映するグラフ** (同じ期間におけるパイプライン実行の合計、平均、および最大継続時間、** 同じ期間におけるパイプライン実行の合計数) の情報が含まれます。

   パイプラインの表も表示されます。この表には、期間内に実行されたすべてのパイプラインがリストされ、その期間と成功率が示されます。
2. オプション: 必要に応じて、統計表示の設定を変更します。
   - **Project**: 統計を表示するプロジェクトまたは namespace。
   - **Time range**: 統計を表示する期間。
   - **Refresh interval**: 表示中に Red Hat OpenShift Pipelines がウィンドウのデータを更新する必要がある頻度。

2.4.3. 特定のパイプラインの統計の表示

特定のパイプラインに関連する統計情報を表示できます。

前提条件

- Red Hat OpenShift Pipelines Operator がクラスターにインストールされている。
- Tekton Results をインストールしている。
- OpenShift Pipelines Web コンソールプラグインがインストールされている。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Pipelines → Pipelines** を選択します。
2. Pipeline リストでパイプラインをクリックします。**Pipeline details** ビューが表示されます。
3. **Metrics** タブをクリックします。
   統計の概要が表示されます。この概要には、**一定期間におけるパイプライン実行の数とステータスを反映するグラフ** (同じ期間におけるパイプライン実行の合計、平均、および最大継続時間、** 同じ期間におけるパイプライン実行の合計数) の情報が含まれます。
4. オプション: 必要に応じて、統計表示の設定を変更します。
   - **Project**: 統計を表示するプロジェクトまたは namespace。
   - **Time range**: 統計を表示する期間。
   - **Refresh interval**: 表示中に Red Hat OpenShift Pipelines がウィンドウのデータを更新する必要がある頻度。

第3章 リゾルバーを使用したリモートパイプラインとタスクの指定

パイプラインとタスクは、CI/CD プロセスの再利用可能なブロックです。以前に開発したパイプラインやタスク、または他の人が開発したパイプラインやタスクを、定義をコピーして貼り付けることなく再利用できます。これらのパイプラインまたはタスクは、クラスター上の他の namespace からパブリックカタログに至るまで、いくつかの種類のソースから利用できます。

パイプライン実行リソースでは、既存のソースからパイプラインを指定できます。パイプラインリソースまたはタスク実行リソースでは、既存のソースからタスクを指定できます。

このような場合、Red Hat OpenShift Pipelines の **リゾルバー** は、実行時に指定されたソースからパイプラインまたはタスク定義を取得します。

以下のリゾルバーは、Red Hat OpenShift Pipelines のデフォルトのインストールで使用できます。

ハブリゾルバー

Artifact Hub または Tekton Hub で利用可能な Pipelines Catalog からタスクまたはパイプラインを取得します。

バンドルリゾルバー

Tekton バンドルからタスクまたはパイプラインを取得します。Tekton バンドルは、OpenShift コンテナーリポジトリーなどの任意の OCI リポジトリーから利用できる OCI イメージです。

クラスターリゾルバー

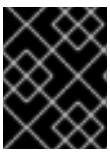
特定の namespace の同じ OpenShift Container Platform クラスター上にすでに作成されているタスクまたはパイプラインを取得します。

Git リゾルバー

Git リポジトリーからタスクまたはパイプラインバインディングを取得します。リポジトリー、ブランチ、パスを指定する必要があります。

3.1. TEKTON カタログからのリモートパイプラインまたはタスクの指定

ハブリゾルバーを使用して、Artifact Hub のパブリック Tekton カタログまたは Tekton Hub のインスタンスで定義されたリモートパイプラインまたはタスクを指定できます。



重要

Artifact Hub プロジェクトは、Red Hat OpenShift Pipelines ではサポートされていません。Artifact Hub の設定のみがサポートされます。

3.1.1. ハブリゾルバーの設定

ハブリゾルバーを設定することで、リソースをプルするためのデフォルトのハブとデフォルトのカタログ設定を変更できます。

手順

1. **TektonConfig** カスタムリソースを編集するには、次のコマンドを入力します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** カスタムリソースで、**pipeline.hub-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    hub-resolver-config:
      default-tekton-hub-catalog: Tekton (1)
      default-artifact-hub-task-catalog: tekton-catalog-tasks (2)
      default-artifact-hub-pipeline-catalog: tekton-catalog-pipelines (3)
      defailt-kind: pipeline (4)
      default-type: tekton (5)
      tekton-hub-api: "https://my-custom-tekton-hub.example.com" (6)
      artifact-hub-api: "https://my-custom-artifact-hub.example.com" (7)
```

1. リソースをプルするためのデフォルトの Tekton Hub カタログ。
2. タスクリソースをプルするためのデフォルトの Artifact Hub カタログ。
3. パイプラインリソースをプルするためのデフォルトの Artifact Hub カタログ。
4. 参照のデフォルトのオブジェクトの種類。
5. リソースをプルするためのデフォルトのハブ。Artifact Hub の場合は **artifact**、Tekton Hub の場合は **tekton**。
6. **default-type** オプションが **tekton** に設定されている場合に使用される Tekton Hub API。
7. オプション: **default-type** オプションが **artifact** に設定されている場合に使用される Artifact Hub API。

重要

default-type オプションを **tekton** に設定する場合は、**tekton-hub-api** 値を設定して Tekton Hub の独自のインスタンスを設定する必要があります。

default-type オプションを **artifact** に設定すると、リゾルバーはデフォルトで https://artifacthub.io/ のパブリックハブ API を使用します。**artifact-hub-api** 値を設定することで、独自の Artifact Hub API を設定できます。

3.1.2. ハブリゾルバーを使用したリモートパイプラインまたはタスクの指定

パイプライン実行を作成するときに、Artifact Hub または Tekton Hub からリモートパイプラインを指定できます。パイプラインまたはタスク実行を作成するときに、Artifact Hub または Tekton Hub からリモートタスクを指定できます。

手順

- Artifact Hub または Tekton Hub からリモートパイプラインまたはタスクを指定するには、**pipelineRef** または **taskRef** 仕様で次の参照形式を使用します。

```
# ...
  resolver: hub
```

```
  params:
  - name: catalog
    value: <catalog>
  - name: type
    value: <catalog_type>
  - name: kind
    value: [pipeline|task]
  - name: name
    value: <resource_name>
  - name: version
    value: <resource_version>
# ...
```

表3.1 ハブリゾルバーでサポートされるパラメーター

パラメーター	Description	値の例
catalog	リソースを取得するためのカタログ。	デフォルト: **tekton-catalog-tasks** (**task** の種類)。**tekton-catalog-pipelines** (**pipeline** の種類の場合)。
type	リソースをプルするカタログのタイプ。Artifact Hub の場合は **Artifact**、Tekton Hub の場合は **tekton** のいずれかです。	デフォルト: **artifact**
kind	**task** または **pipeline** のいずれか。	デフォルト: **task**
name	ハブから取得するタスクまたはパイプラインの名前。	**golang-build**
version	ハブから取得するタスクまたはパイプラインのバージョン。数値を引用符 (**''**) で囲む必要があります。	**"0.5.0"**

パイプラインまたはタスクに追加のパラメーターが必要な場合は、これらのパラメーターを **params** に指定します。

次のパイプライン実行の例では、カタログからリモートパイプラインを参照します。

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: hub-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: hub
    params:
    - name: catalog
      value: tekton-catalog-pipelines
```

```
    - name: type
      value: artifact
    - name: kind
      value: pipeline
    - name: name
      value: example-pipeline
    - name: version
      value: "0.1"
    - name: sample-pipeline-parameter
      value: test
```

次のパイプラインの例は、カタログからリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
  - name: "cluster-task-reference-demo"
    taskRef:
      resolver: hub
      params:
      - name: catalog
        value: tekton-catalog-tasks
      - name: type
        value: artifact
      - name: kind
        value: task
      - name: name
        value: example-task
      - name: version
        value: "0.6"
      - name: sample-task-parameter
        value: test
```

次のタスク実行例では、カタログからリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: hub
    params:
    - name: catalog
      value: tekton-catalog-tasks
    - name: type
      value: artifact
    - name: kind
      value: task
    - name: name
      value: example-task
    - name: version
```

```
    value: "0.6"
  - name: sample-task-parameter
    value: test
```

3.2. TEKTON バンドルからのリモートパイプラインまたはタスクの指定

バンドルリゾルバーを使用して、Tekton バンドルからリモートパイプラインまたはタスクを指定できます。Tekton バンドルは、OpenShift コンテナーリポジトリーなどの任意の OCI リポジトリーから利用できる OCI イメージです。

3.2.1. バンドルリゾルバーの設定

バンドルリゾルバーを設定することで、Tekton バンドルからリソースを取得するためのデフォルトのサービスアカウント名とデフォルトの種類を変更できます。

手順

1. **TektonConfig** カスタムリソースを編集するには、次のコマンドを入力します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** カスタムリソースで、**pipeline.bundles-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines 1
      default-kind: task 2
```

1. バンドルリクエストに使用するデフォルトのサービスアカウント名。
2. バンドルイメージのデフォルトレイヤーの種類。

3.2.2. バンドルリゾルバーを使用したリモートパイプラインまたはタスクの指定

パイプライン実行を作成するときに、Tekton バンドルからリモートパイプラインを指定できます。パイプラインまたはタスク実行を作成するときに、Tekton バンドルからリモートタスクを指定できます。

手順

- Tekton バンドルからリモートパイプラインまたはタスクを指定するには、**pipelineRef** または **taskRef** 仕様で次の参照形式を使用します。

```
# ...
  resolver: bundles
  params:
  - name: bundle
```

```
    value: <fully_qualified_image_name>
  - name: name
    value: <resource_name>
  - name: kind
    value: [pipeline|task]
# ...
```

表3.2 バンドルリゾルバーでサポートされているパラメーター

パラメーター	Description	値の例
serviceAccount	レジストリー認証情報を作成するときに使用するサービスアカウントの名前。	**default**
bundle	取得するイメージを指すバンドル URL。	**gcr.io/tekton-releases/catalog/upstream/golang-build:0.1**
name	バンドルから取り出すリソースの名前。	**golang-build**
kind	バンドルから取り出すリソースの種類。	**task**

パイプラインまたはタスクに追加のパラメーターが必要な場合は、これらのパラメーターを **params** に指定します。

次のパイプライン実行の例は、Tekton バンドルからのリモートパイプラインを参照します。

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: bundle-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: bundles
    params:
    - name: bundle
      value: registry.example.com:5000/simple/pipeline:latest
    - name: name
      value: hello-pipeline
    - name: kind
      value: pipeline
    - name: sample-pipeline-parameter
      value: test
  params:
  - name: username
    value: "pipelines"
```

次のパイプラインの例は、Tekton バンドルからのリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
```

```
kind: Pipeline
metadata:
  name: pipeline-with-bundle-task-reference-demo
spec:
  tasks:
  - name: "bundle-task-demo"
    taskRef:
      resolver: bundles
      params:
      - name: bundle
        value: registry.example.com:5000/advanced/task:latest
      - name: name
        value: hello-world
      - name: kind
        value: task
      - name: sample-task-parameter
        value: test
```

次のタスク実行例では、Tekton バンドルのリモートタスクを参照しています。

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: bundle-task-reference-demo
spec:
  taskRef:
    resolver: bundles
    params:
    - name: bundle
      value: registry.example.com:5000/simple/new_task:latest
    - name: name
      value: hello-world
    - name: kind
      value: task
    - name: sample-task-parameter
      value: test
```

3.3. 同じクラスターからのリモートパイプラインまたはタスクの指定

クラスターリゾルバーを使用して、Red Hat OpenShift Pipelines が実行している OpenShift Container Platform クラスター上の namespace で定義されているリモートパイプラインまたはタスクを指定できます。

3.3.1. クラスターリゾルバーの設定

クラスターリゾルバーのデフォルトの種類と namesspace を変更したり、クラスターリゾルバーが使用できる名前空間を制限したりできます。

手順

1. **TektonConfig** カスタムリソースを編集するには、次のコマンドを入力します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** カスタムリソースで、**pipeline.cluster-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    cluster-resolver-config:
      default-kind: pipeline (1)
      default-namespace: namespace1 (2)
      allowed-namespaces: namespace1, namespace2 (3)
      blocked-namespaces: namespace3, namespace4 (4)
```

(1) パラメーターで指定されていない場合は、フェッチするデフォルトのリソースの種類。

(2) パラメーターで指定されていない場合、リソースを取得するためのデフォルトの namespace。

(3) リゾルバーがアクセスを許可される namespace のコンマ区切りのリスト。このキーが定義されていない場合は、すべての namespace が許可されます。

(4) リゾルバーがアクセスをブロックされる namespace のオプションのコンマ区切りのリスト。このキーが定義されていない場合は、すべての namespace が許可されます。

3.3.2. クラスターリゾルバーを使用したリモートパイプラインまたはタスクの指定

パイプライン実行を作成するときに、同じクラスターからのリモートパイプラインを指定できます。パイプラインまたはタスク実行を作成するときに、同じクラスターからリモートタスクを指定できます。

手順

- 同じクラスターからリモートパイプラインまたはタスクを指定するには、**pipelineRef** または **taskRef** 仕様で以下の参照形式を使用します。

```
# ...
  resolver: cluster
  params:
  - name: name
    value: <name>
  - name: namespace
    value: <namespace>
  - name: kind
    value: [pipeline|task]
# ...
```

表3.3 クラスターリゾルバーでサポートされているパラメーター

パラメーター	Description	値の例
name	取得するリソースの名前。	**some-pipeline**

パラメーター	Description	値の例
namespace	リソースを含むクラスター内の名前空間。	**other-namespace**
kind	取得するリソースの種類。	**パイプライン**

パイプラインまたはタスクに追加のパラメーターが必要な場合は、これらのパラメーターを **params** に指定します。

次のパイプライン実行の例は、同じクラスターからのリモートパイプラインを参照します。

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: cluster-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: cluster
    params:
    - name: name
      value: some-pipeline
    - name: namespace
      value: test-namespace
    - name: kind
      value: pipeline
    - name: sample-pipeline-parameter
      value: test
```

次のパイプラインの例は、同じクラスターのリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
  - name: "cluster-task-reference-demo"
    taskRef:
      resolver: cluster
      params:
      - name: name
        value: some-task
      - name: namespace
        value: test-namespace
      - name: kind
        value: task
      - name: sample-task-parameter
        value: test
```

次のタスク実行例では、同じクラスターのリモートタスクを参照しています。

```
apiVersion: tekton.dev/v1
```

```
kind: TaskRun
metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: cluster
    params:
    - name: name
      value: some-task
    - name: namespace
      value: test-namespace
    - name: kind
      value: task
    - name: sample-task-parameter
      value: test
```

3.4. GIT リポジトリーからのリモートパイプラインまたはタスクの指定

Git リゾルバーを使用して、Git リポジトリーからリモートパイプラインまたはタスクを指定できます。リポジトリーには、パイプラインまたはタスクを定義する YAML ファイルが含まれている必要があります。Git リゾルバーは、匿名でクローンを作成するか、認証された SCM API を使用してリポジトリーにアクセスできます。

3.4.1. 匿名 Git クローン作成用の Git リゾルバーの設定

匿名 Git クローン作成を使用する場合は、Git リポジトリーからリモートパイプラインとタスクをプルするためのデフォルトの Git リビジョン、フェッチタイムアウト、およびデフォルトのリポジトリー URL を設定できます。

手順

1. **TektonConfig** カスタムリソースを編集するには、次のコマンドを入力します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** カスタムリソースで、**pipeline.git-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
      default-revision: main (1)
      fetch-timeout: 1m (2)
      default-url: https://github.com/tektoncd/catalog.git (3)
```

(1) 何も指定されていない場合に使用するデフォルトの Git リビジョン。

(2) 単一の Git クローン解決にかかる最大時間は、たとえば、**1m**、**2s**、**700ms** です。Red Hat OpenShift Pipelines は、すべての解決リクエストに対して 1 分のグローバル最大タイムアウトも適用します。

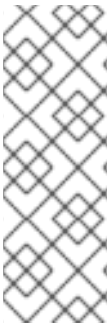
3 何も指定されていない場合、匿名クローン作成用のデフォルトの Git リポジトリー URL。

3.4.2. 認証された SCM API の Git リゾルバーの設定

認証された SCM API の場合、認証された Git 接続の設定を設定する必要があります。

go-scm ライブラリーでサポートされている Git リポジトリープロバイダーを使用できます。すべての **go-scm** 実装が Git リゾルバーでテストされているわけではありませんが、次のプロバイダーが動作することが確認されています。

- **GitHub.com** および GitHub Enterprise
- **GitLab.com** およびセルフホスト Gitlab
- Gitea
- BitBucket Server
- BitBucket Cloud



注記

- クラスターの認証済み SCM API を使用して設定できる Git 接続は 1 つだけです。この接続は、クラスターのすべてのユーザーが利用できるようになります。クラスターのすべてのユーザーは、接続用に設定したセキュリティートークンを使用してリポジトリーにアクセスできます。
- 認証された SCM API を使用するように Git リゾルバーを設定すると、匿名の Git クローン参照を使用してパイプラインとタスクを取得することもできます。

手順

1. **TektonConfig** カスタムリソースを編集するには、次のコマンドを入力します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** カスタムリソースで、**pipeline.git-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
      default-revision: main 1
      fetch-timeout: 1m 2
      scm-type: github 3
      server-url: api.internal-github.com 4
      api-token-secret-name: github-auth-secret 5
      api-token-secret-key: github-auth-key 6
      api-token-secret-namespace: github-auth-namespace 7
      default-org: tektoncd 8
```

1. 何も指定されていない場合に使用するデフォルトの Git リビジョン。
2. 単一の Git クローン解決にかかる最大時間は、たとえば、**1m**、**2s**、**700ms** です。Red Hat OpenShift Pipelines は、すべての解決リクエストに対して 1 分のグローバル最大タイムアウトも適用します。
3. SCM プロバイダーのタイプ。
4. 認証された SCM API で使用するベース URL。**github.com**、**gitlab.com**、または BitBucket Cloud を使用している場合、この設定は必要ありません。
5. SCM プロバイダー API トークンを含むシークレットの名前。
6. トークンを含むトークンシークレット内のキー。
7. トークンシークレットを含む名前空間 (**default** でない場合)。
8. オプション: 認証された API を使用する場合のリポジトリーのデフォルトの組織。この組織は、リゾルバーパラメーターで組織を指定しない場合に使用されます。

注記

認証された SCM API を使用するには、**scm-type**、**api-token-secret-name**、および **api-token-secret-key** 設定が必要です。

3.4.3. Git リゾルバーを使用したリモートパイプラインまたはタスクの指定

パイプライン実行を作成するときに、Git リポジトリーからリモートパイプラインを指定できます。パイプラインまたはタスク実行を作成するときに、Git リポジトリーからリモートタスクを指定できます。

前提条件

- 認証された SCM API を使用する場合は、Git リゾルバーに対して認証された Git 接続を設定する必要があります。

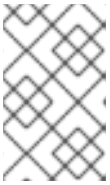
手順

1. Git リポジトリーからリモートパイプラインまたはタスクを指定するには、**pipelineRef** または **taskRef** 仕様で次の参照形式を使用します。

```
# ...
  resolver: git
  params:
  - name: url
    value: <git_repository_url>
  - name: revision
    value: <branch_name>
  - name: pathInRepo
    value: <path_in_repository>
# ...
```

表3.4 Git リゾルバーでサポートされているパラメーター

パラメーター	Description	値の例
url	匿名クローン作成を使用する場合のリポジトリーの URL。	**https://github.com/tektoncd/catalog.git**
repo	認証された SCM API を使用する場合のリポジトリー名。	**test-infra**
org	認証された SCM API を使用する場合のリポジトリーの組織。	**tektoncd**
revision	リポジトリー内の Git リビジョン。ブランチ名、タグ名、またはコミット SHA ハッシュを指定できます。	**aeb957601cf41c012be462827053a21a420befca** **main** **v0.38.2**
pathInRepo	リポジトリー内の YAML ファイルのパス名。	**task/golang-build/0.3/golang-build.yaml**



注記

リポジトリーのクローンを作成して匿名で取得するには、**url** パラメーターを使用します。認証された SCM API を使用するには、**repo** パラメーターを使用します。**url** パラメーターと **repo** パラメーターを同時に指定しないでください。

パイプラインまたはタスクに追加のパラメーターが必要な場合は、これらのパラメーターを **params** に指定します。

次のパイプライン実行の例では、Git リポジトリーからリモートパイプラインを参照します。

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: git-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: git
    params:
    - name: url
      value: https://github.com/tektoncd/catalog.git
    - name: revision
      value: main
    - name: pathInRepo
      value: pipeline/simple/0.1/simple.yaml
    - name: sample-pipeline-parameter
      value: test
  params:
  - name: name
    value: "testPipelineRun"
```

次のパイプラインの例では、Git リポジトリーからリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-git-task-reference-demo
spec:
  tasks:
  - name: "git-task-reference-demo"
    taskRef:
      resolver: git
      params:
      - name: url
        value: https://github.com/tektoncd/catalog.git
      - name: revision
        value: main
      - name: pathInRepo
        value: task/git-clone/0.6/git-clone.yaml
      - name: sample-task-parameter
        value: test
```

次のタスク実行例では、Git リポジトリーからリモートタスクを参照します。

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: git-task-reference-demo
spec:
  taskRef:
    resolver: git
    params:
    - name: url
      value: https://github.com/tektoncd/catalog.git
    - name: revision
      value: main
    - name: pathInRepo
      value: task/git-clone/0.6/git-clone.yaml
    - name: sample-task-parameter
      value: test
```

3.5. 関連情報

- OpenShift Pipelines での Tekton Hub の使用

第4章 パイプラインでの RED HAT エンタイトルメントの使用

Red Hat Enterprise Linux (RHEL) エンタイトルメントをお持ちの場合は、これらのエンタイトルメントを使用してパイプライン内にコンテナーイメージを構築できます。

Insight Operator は、Simple Common Access (SCA) からこの Operator にエンタイトルメントをインポートした後、エンタイトルメントを自動的に管理します。この Operator は、**openshift-config-managed** namespace に **etc-pki-entitlement** という名前のシークレットを提供します。

次の 2 つの方法のいずれかで、パイプラインで Red Hat エンタイトルメントを使用できます。

- シークレットをパイプラインの namespace に手動でコピーします。パイプライン namespace の数が限られている場合、この方法が最も複雑性が低くなっています。
- Shared Resources Container Storage Interface (CSI) Driver Operator を使用して、namespace 間でシークレットを自動的に共有します。

4.1. 前提条件

- **oc** コマンドラインツールを使用して OpenShift Container Platform クラスターにログインしている。
- OpenShift Container Platform クラスターで Insights Operator 機能を有効にしている。Shared Resources CSI Driver Operator を使用して namespace 間でシークレットを共有する場合は、Shared Resources CSI ドライバーも有効にする必要があります。Insights Operator や Shared Resources CSI Driver などの機能を有効にする方法については、フィーチャーゲートを使用した機能の有効化 を参照してください。

注記

Insights Operator を有効にした後、クラスターがこの Operator を使用してすべてのノードを更新するまで、しばらく待つ必要があります。次のコマンドを入力すると、すべてのノードのステータスを監視できます。

```
$ oc get nodes -w
```

Insights Operator がアクティブであることを確認するには、次のコマンドを入力して、**insights-operator** Pod が **openshift-insights** namespace で実行されていることを確認します。

```
$ oc get pods -n openshift-insights
```

- Red Hat エンタイトルメントの Insights Operator へのインポートを設定しました。エンタイトルメントのインポートに関する詳細は、Insights Operator を使用した Simple Content Access エンタイトルメントのインポート を参照してください。

注記

Insights Operator がエンタイトルメントを利用可能にし、アクティブであることを確認するには、以下のコマンドを入力して、**etc-pki-entitlement** シークレットが **openshift-config-managed** namespace に存在することを確認します。

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed
```

4.2. ETC-PKI-ENTITLEMENT シークレットの手動コピーによる RED HAT エンタイトルメントの使用

etc-pki-entitlement シークレットを **openshift-config-managed** namespace からパイプラインの namespace にコピーできます。次に、Buildah タスクにこのシークレットを使用するようにパイプラインを設定できます。

前提条件

- システムに **jq** パッケージをインストールしている。このパッケージは Red Hat Enterprise Linux (RHEL) で利用できます。

手順

1. 次のコマンドを実行して、**etc-pki-entitlement** シークレットを **openshift-config-managed** namespace からパイプラインの namespace にコピーします。

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed -o json | \
  jq 'del(.metadata.resourceVersion)' | jq 'del(.metadata.creationTimestamp)' | \
  jq 'del(.metadata.uid)' | jq 'del(.metadata.namespace)' | \
  oc -n <pipeline_namespace> create -f - 1
```

1 **<pipeline_namespace>** は、パイプラインの namespace に置き換えます。

2. Buildah タスク定義では、次の例のように、**buildah** クラスタータスクまたはこのクラスタータスクのコピーを使用して、**rhel-entitlement** ワークスペースを定義します。

3. タスク実行または Buildah タスクを実行するパイプライン実行で、次の例のように、**etc-pki-entitlement** シークレットを **rhel-entitlement** ワークスペースに割り当てます。

Red Hat エンタイトルメントを使用するパイプライン実行定義の例 (パイプラインとタスクの定義を含む)

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement 1
      secret:
        secretName: etc-pki-entitlement
```

```
  pipelineSpec:
    workspaces:
      - name: shared-workspace
      - name: dockerconfig
      - name: rhel-entitlement 2
    tasks:
# ...
      - name: buildah
        taskRef:
          name: buildah
          kind: ClusterTask
        workspaces:
        - name: source
          workspace: shared-workspace
        - name: dockerconfig
          workspace: dockerconfig
        - name: rhel-entitlement 3
          workspace: rhel-entitlement
        params:
        - name: IMAGE
          value: <image_where_you_want_to_push>
```

1. パイプライン実行での **rhel-entitlement** ワークスペースの定義 (ワークスペースに **etc-pki-entitlement** シークレットを割り当てます)
2. パイプライン定義の **rhel-entitlement** ワークスペースの定義
3. タスク定義の **rhel-entitlement** ワークスペースの定義

4.3. 共有リソース CSI ドライバー OPERATOR を使用してシークレットを共有することによる RED HAT エンタイトルメントの使用

Shared Resources Container Storage Interface (CSI) Driver Operator を使用して、**openshift-config-managed** namespace から他の namespace への **etc-pki-entitlement** シークレットの共有を設定できます。次に、Buildah タスクにこのシークレットを使用するようにパイプラインを設定できます。

前提条件

- クラスター管理者のパーミッションを持つユーザーとして **oc** コマンドラインユーティリティーを使用して、OpenShift Container Platform クラスターにログインしている。
- OpenShift Container Platform クラスターで Shared Resources CSI Driver Operator を有効にしている。

手順

1. 次のコマンドを実行して、**etc-pki-entitlement** シークレットを共有するための **SharedSecret** カスタムリソース (CR) を作成します。

```
$ oc apply -f - <<EOF
apiVersion: sharedresource.openshift.io/v1alpha1
kind: SharedSecret
metadata:
```

```
  name: shared-rhel-entitlement
spec:
  secretRef:
    name: etc-pki-entitlement
    namespace: openshift-config-managed
EOF
```

2. 次のコマンドを実行して、共有シークレットへのアクセスを許可する RBAC ロールを作成します。

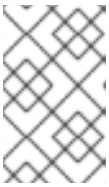
```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-rhel-entitlement
  namespace: <pipeline_namespace> 1
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedsecrets
    resourceNames:
      - shared-rhel-entitlement
    verbs:
      - use
EOF
```

1 **<pipeline_namespace>** は、パイプラインの namespace に置き換えます。

3. 次のコマンドを実行して、**pipeline** サービスアカウントにロールを割り当てます。

```
$ oc create rolebinding shared-resource-rhel-entitlement --role=shared-shared-resource-rhel-entitlement \
  --serviceaccount=<pipeline-namespace>:pipeline 1
```

1 **<pipeline-namespace>** をパイプラインの namespace に置き換えます。



注記

OpenShift Pipelines のデフォルトのサービスアカウントを変更した場合、またはパイプライン実行またはタスク実行でカスタムサービスアカウントを定義した場合は、**pipeline** アカウントではなくこのアカウントにロールを割り当てます。

4. Buildah タスク定義では、次の例のように、**buildah** クラスタータスクまたはこのクラスタータスクのコピーを使用して、**rhel-entitlement** ワークスペースを定義します。

5. タスク実行または Buildah タスクを実行するパイプライン実行で、次の例のように、共有シークレットを **rhel-entitlement** ワークスペースに割り当てます。

Red Hat エンタイトルメントを使用するパイプライン実行定義の例 (パイプラインとタスクの定義を含む)

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test-csi
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement (1)
      csi:
        readOnly: true
        driver: csi.sharedresource.openshift.io
        volumeAttributes:
          sharedSecret: shared-rhel-entitlement
  pipelineSpec:
    workspaces:
      - name: shared-workspace
      - name: dockerconfig
      - name: rhel-entitlement (2)
    tasks:
# ...
      - name: buildah
        taskRef:
          name: buildah
          kind: ClusterTask
        workspaces:
        - name: source
          workspace: shared-workspace
        - name: dockerconfig
          workspace: dockerconfig
        - name: rhel-entitlement (3)
          workspace: rhel-entitlement
        params:
        - name: IMAGE
          value: <image_where_you_want_to_push>
```

(1) パイプラインでの **rhel-entitlement** ワークスペースの定義が実行され、**shared-rhel-entitlement** CSI 共有シークレットがワークスペースに割り当てられます。

(2) パイプライン定義の **rhel-entitlement** ワークスペースの定義

(3) タスク定義の **rhel-entitlement** ワークスペースの定義

4.4. 関連情報

- Simple content access
- Insights Operator の使用
- Insights Operator を使用した Simple Content Access エンタイトルメントのインポート
- Shared Resource CSI Driver Operator
- OpenShift Pipelines のデフォルトサービスアカウントの変更

第5章 バージョン管理されていないクラスタータスクおよびバージョン管理されたクラスタータスクの管理

クラスター管理者は、Red Hat OpenShift Pipelines Operator をインストールすると、**バージョン付けされたクラスタータスク**(VCT) および**バージョン付けされていないクラスタータスク** (NVCT) として知られるそれぞれのデフォルトクラスタータスクのバリアントが作成されます。たとえば、Red Hat OpenShift Pipelines Operator v1.7 をインストールすると、**buildah-1-7-0** VCT および **buildah** NVCT が作成されます。

NVCT と VCT の両方は、**params**、**workspaces**、および **steps** など、同じメタデータ、動作、仕様を持ちます。ただし、それらを無効にするか、Operator をアップグレードすると、動作が異なります。

重要

Red Hat OpenShift Pipelines 1.10 では、クラスタータスク機能は非推奨であり、将来のリリースで削除される予定です。

5.1. バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの違い

バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクでは、命名規則が異なります。また、Red Hat OpenShift Pipelines Operator はそれらを異なる方法でアップグレードします。

表5.1 バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの違い

	バージョン付けされていないクラスタータスク	バージョン付けされたクラスタータスク
命名法	NVCT には、クラスタータスクの名前のみが含まれます。たとえば、Operator v1.7 でインストールされた Buildah の NVCT の名前は **buildah** です。	VCT には、クラスタータスクの名前の後にバージョンが接尾辞として含まれます。たとえば、Operator v1.7 でインストールされた Buildah の VCT の名前は **buildah-1-7-0** です。
アップグレード	Operator をアップグレードすると、最新の変更でバージョン付けされていないクラスタータスクを更新します。NVCT の名前は変更されません。	Operator をアップグレードすると、最新バージョンの VCT をインストールし、以前のバージョンを保持します。VCT の最新バージョンは、アップグレードされた Operator に対応します。たとえば、Operator 1.7 をインストールすると **buildah-1-7-0** がインストールされ、**buildah-1-6-0** は保持されます。

5.2. バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの長所と短所

バージョン付けされていないクラスタータスクまたはバージョン付けされたクラスタータスクを実稼働環境で標準として導入する前に、クラスター管理者はその長所と短所を検討する場合があります。

表5.2 バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの長所と短所

クラスタータスク	メリット	デメリット
バージョン付けされていないクラスタータスク (NVCT)	• 最新の更新およびバグ修正でパイプラインをデプロイする場合は、NVCT を使用します。 • Operator をアップグレードすると、バージョン付けされていないクラスタータスクがアップグレードされます。これは、複数のバージョン付けされたクラスタータスクよりも少ないリソースを消費します。	NVCT を使用するパイプラインをデプロイする場合、自動的にアップグレードされたクラスタータスクが後方互換性を持たない場合、Operator のアップグレード後にそれらが破損する可能性があります。
バージョン付けされたクラスタータスク (VCT)	• 実稼働で安定したパイプラインが重要視される場合は、VCT を使用します。 • 新しいバージョンのクラスタータスクがインストールされた後でも、以前のバージョンはクラスターで保持されます。以前のクラスタータスクを引き続き使用できます。	• 以前のバージョンのクラスタータスクを引き続き使用する場合は、最新の機能と重要なセキュリティー更新が欠落している可能性があります。 • 動作していない以前のバージョンのクラスタータスクがクラスターリソースを消費します。 • * アップグレード後に、Operator は以前の VCT を管理できません。**oc delete clustertask** コマンドを使用して、以前の VCT を手動で削除できますが、復元することはできません。

5.3. バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの無効化

クラスター管理者は、OpenShift Pipeline Operator がインストールしたクラスタータスクを無効にできます。

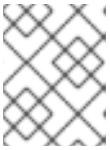
手順

1. バージョン付けされていないクラスタータスクおよび最新のバージョン付けされたクラスタータスクをすべて削除するには、**TektonConfig** カスタムリソース定義 (CRD) を編集し、**spec.addon.params** の **clusterTasks** パラメーターを **false** に設定します。

TektonConfig CR の例

```
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"
...
```

クラスタータスクを無効にすると、Operator はすべてのバージョン付けされていないクラスタータスクおよび最新バージョンのバージョン付けされたクラスタータスクだけをクラスターから削除します。



注記

クラスタータスクを再度有効にすると、バージョン付けされていないクラスタータスクがインストールされます。

2. オプション: バージョン付けされたクラスタータスクの以前のバージョンを削除するには、以下のいずれかの方法を使用します。

   a. 以前のバージョン付けされたクラスタータスクを個別に削除するには、**oc delete clustertask** コマンドの後にバージョン付けされたクラスタータスクの名前を使用します。以下に例を示します。

   ```
   $ oc delete clustertask buildah-1-6-0
   ```

   b. 以前のバージョンの Operator によって作成されたバージョン付けされたクラスタータスクをすべて削除するには、対応するインストーラーセットを削除できます。以下に例を示します。

   ```
   $ oc delete tektoninstallerset versioned-clustertask-1-6-k98as
   ```

   注意

   古いバージョン付けされたクラスタータスクを削除する場合は、これを復元できません。Operator の現行バージョンが作成したバージョン付けされたクラスタータスクおよびバージョン付けされていないクラスタータスクのみを復元できます。