Red Hat OpenShift Pipelines 1.19

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: 2025-07-16

Red Hat OpenShift Pipelines 1.19 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を説明します。この例では、以下で構成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 に一覧表示されている 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 をインストールしている。
- 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-1.19/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.19/01_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
```

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** に対し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。
- **openshift-pipelines** namespace で提供される **buildah** 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-1.19"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
```

```
    - name: namespace
      value: openshift-pipelines
  workspaces:
  - name: output
    workspace: shared-workspace
  params:
  - name: URL
    value: $(params.git-url)
  - name: SUBDIRECTORY
    value: ""
  - name: DELETE_EXISTING
    value: "true"
  - name: REVISION
    value: $(params.git-revision)
- name: build-image
  taskRef:
    resolver: cluster
    params:
    - name: kind
      value: task
    - name: name
      value: buildah
    - name: namespace
      value: openshift-pipelines
  workspaces:
  - name: source
    workspace: shared-workspace
  params:
  - name: IMAGE
    value: $(params.IMAGE)
  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。その詳細は、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-1.19/01_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を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
- 非接続インストールミラーリングについて

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対して **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-1.19/01_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-1.19/01_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

- シークレットを使用したリポジトリーでの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-1.19/03_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-1.19
  - 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-
1.19/03_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:
  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-1.19/03_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:
  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:
  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-1.19/03_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は、このフィールドを省略できます。

他のオプションは、**oc create route reencrypt --help** を参照してください。

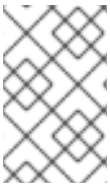
- 非セキュアな HTTP 接続を使用して **EventListener** リソースを作成するには、以下を実行します。

  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** です。

   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が受信する 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** は **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** に対して手順 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、公開される **EventListener** サービスルートにイベントペイロードを送信します。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-1.19
```

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-1.19
```

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は、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は、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は、**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。

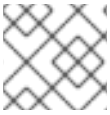
```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
    - name: github-listener
```

```
    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は、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は、**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には、About Pipelines as Code 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 ています。
- 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タスクがクラスターリゾルバーで使用できるクラスターに追加されます。
- 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、**関連情報** 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。ローカルの 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、namespace タスクとパブリック Tekton Hub タスクにのみアクセスできます。

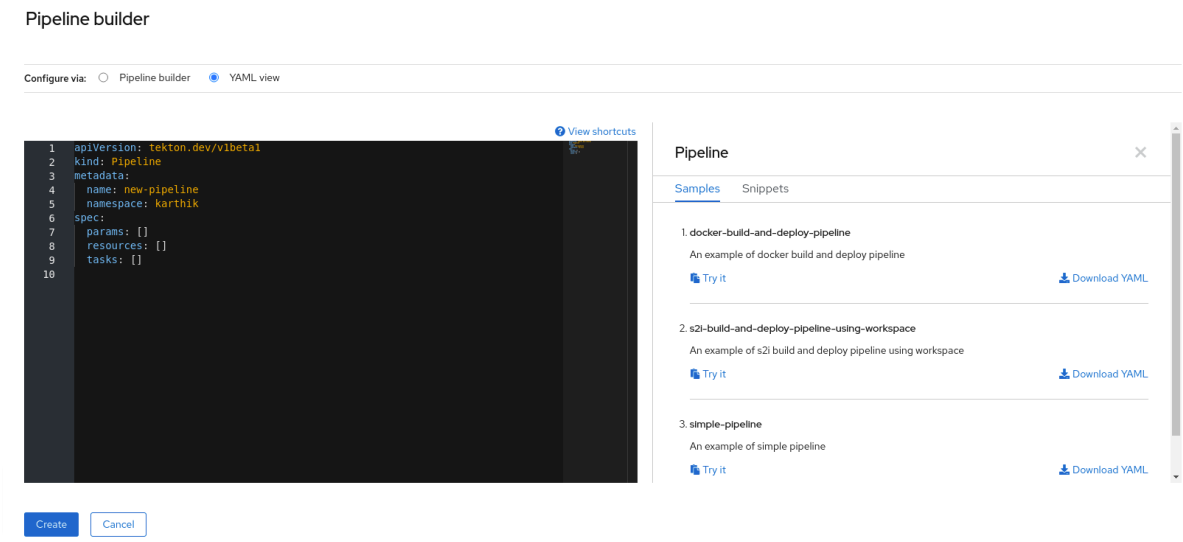
手順

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。

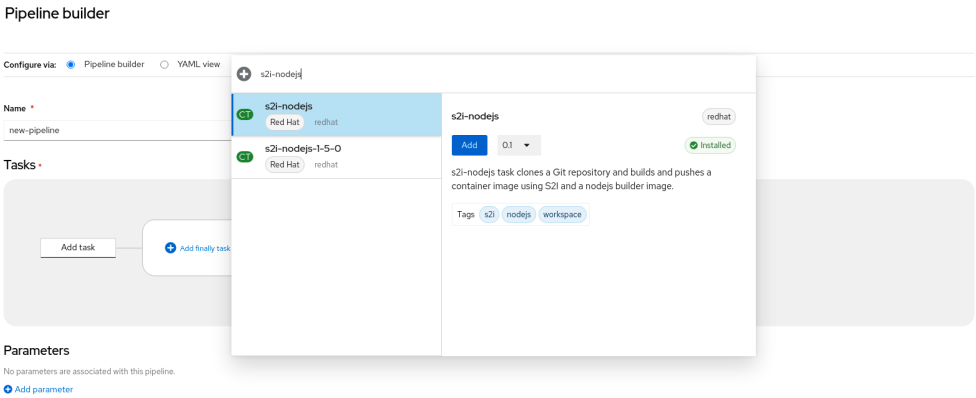
iii. **Add** または **Install and add** をクリックします。この例では、**s2i-nodejs** タスクを使用します。

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 **Image** 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さらにパラメーターを追加します。

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ともに OpenShift Pipeline 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

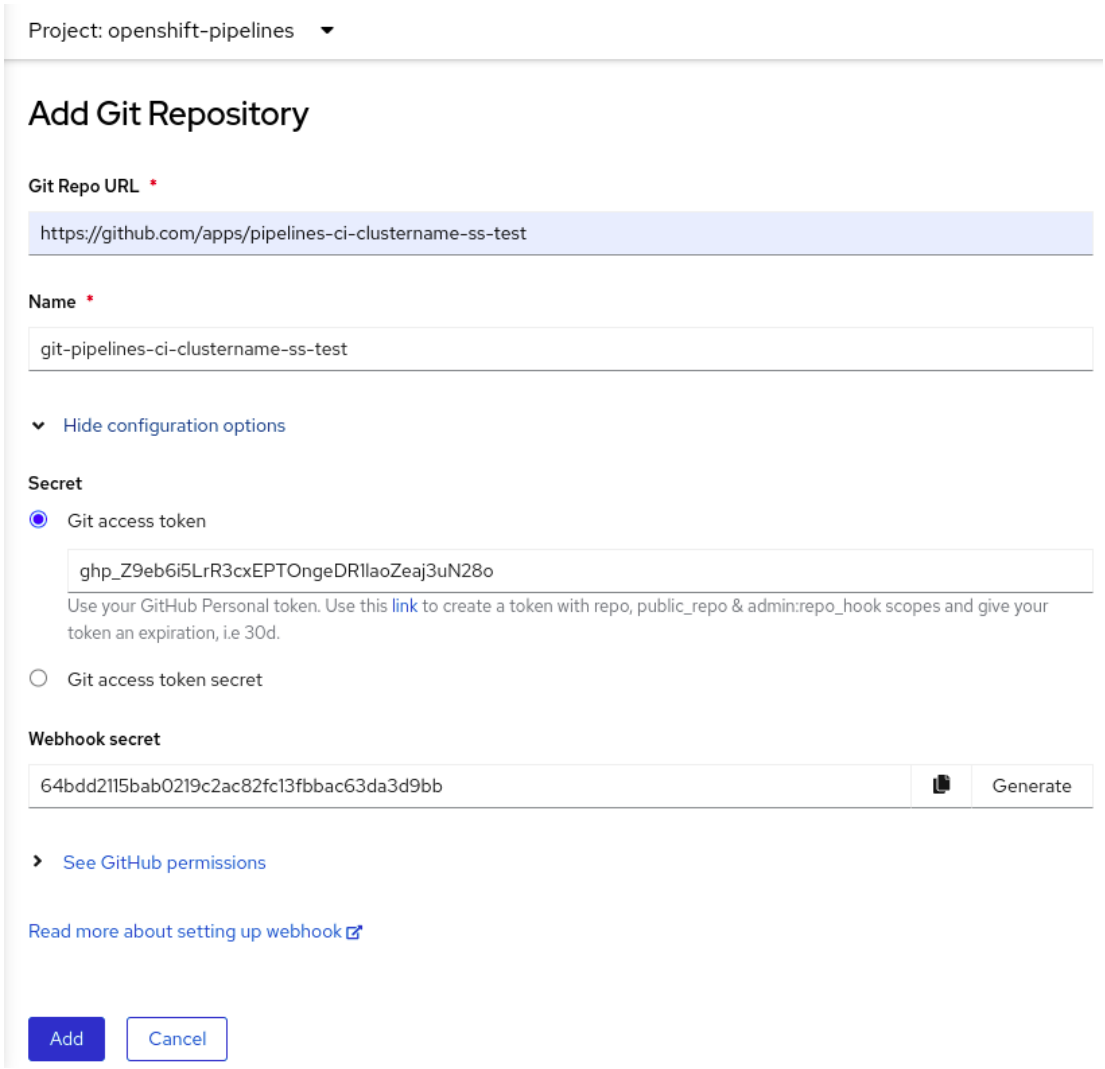
パブリックおよびプライベートの GitHub リポジトリーの両方を追加できます。

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。

手順

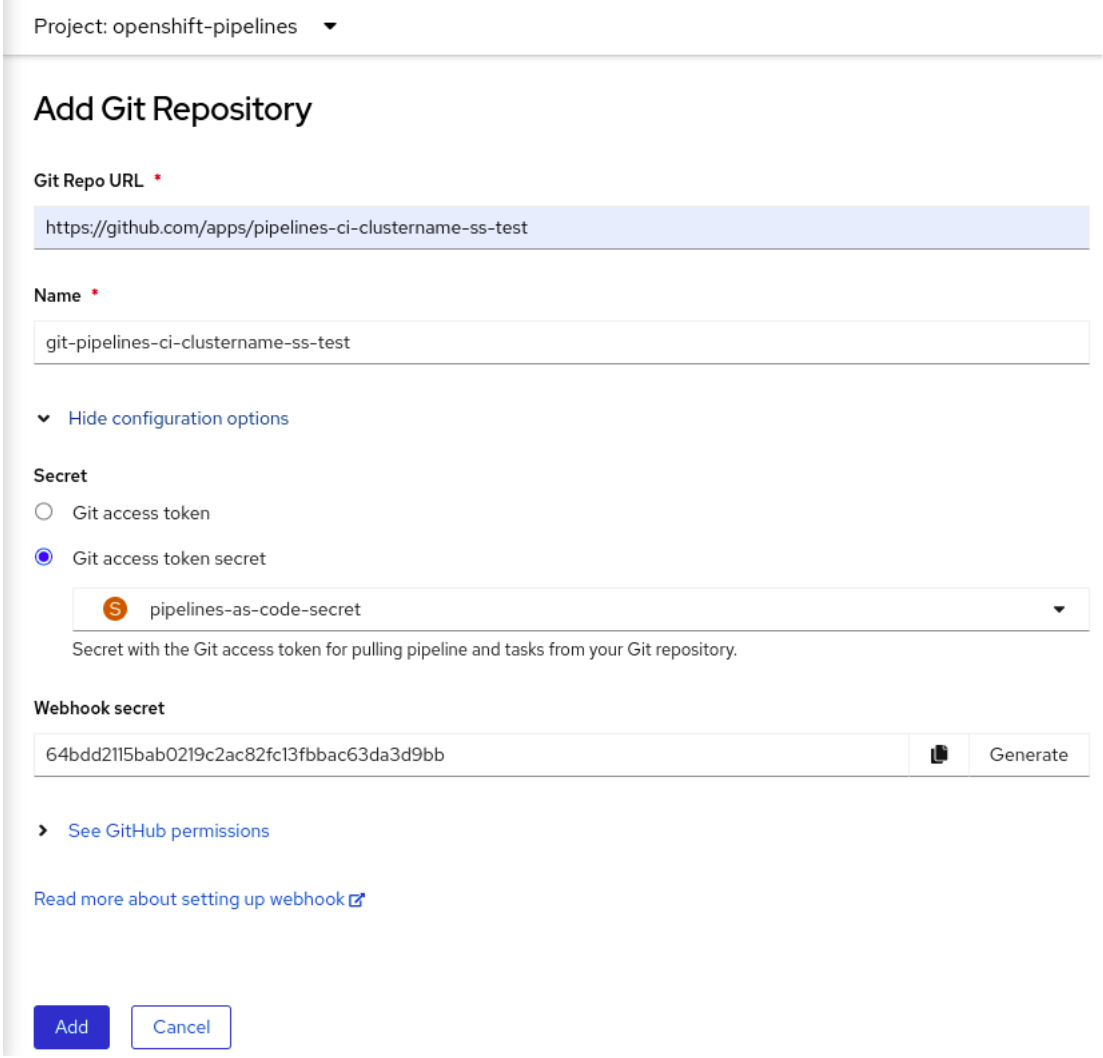
1. **Developer** パースペクティブで、GitHub リポジトリーを追加する namespace またはプロジェクトを選択します。
2. 左側のナビゲーションペインを使用して **Pipelines** に移動します。
3. **Pipelines** ページの右側にある **Create → Repository** をクリックします。
4. **Git Repo URL** を入力すると、コンソールが自動的にリポジトリー名を取得します。
5. **設定オプションを表示** をクリックします。デフォルトでは、**Setup a webhook** というオプションが 1 つだけ表示されます。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は、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

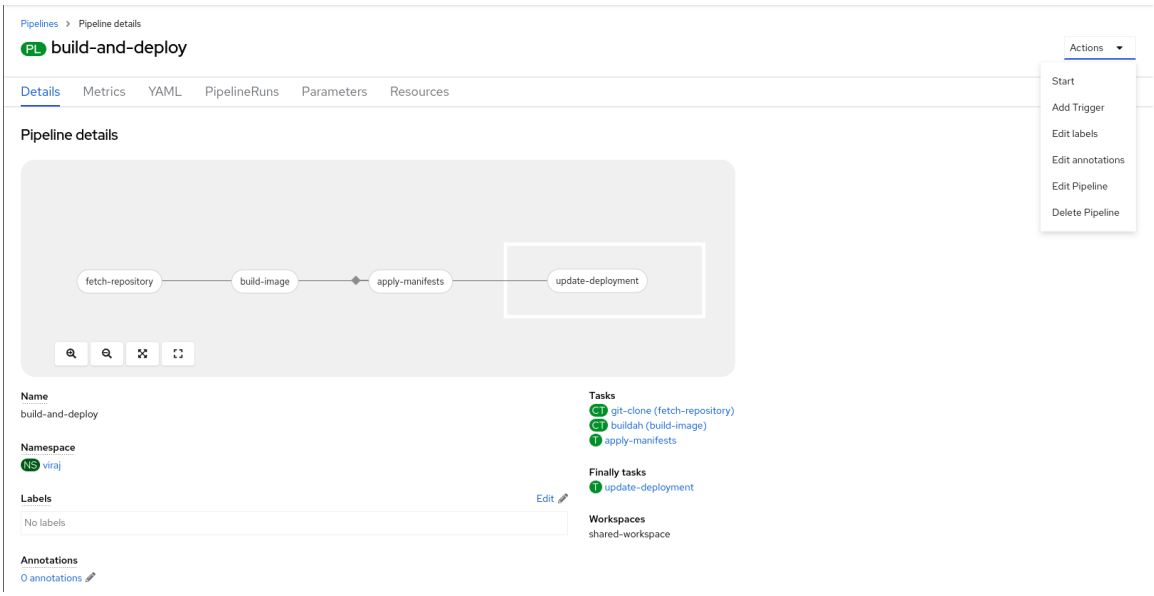
手順

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 Duration**
     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。

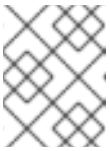
5. オプション: **Pipeline Runs** タブをクリックして、パイプラインの完了済み、実行中、または失敗した実行を確認します。
   Pipeline Runs タブでは、パイプライン実行、タスクのステータス、および失敗したパイプライン実行のデバッグ用のリンクの詳細が表示されます。Options メニュー を使用して、実行中のパイプラインを停止するか、以前のパイプライン実行と同じパラメーターとリソースを使用してパイプラインを再実行するか、パイプライン実行を削除します。

   - 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 ビューからのパイプラインの開始

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は、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は、**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、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、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、**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。

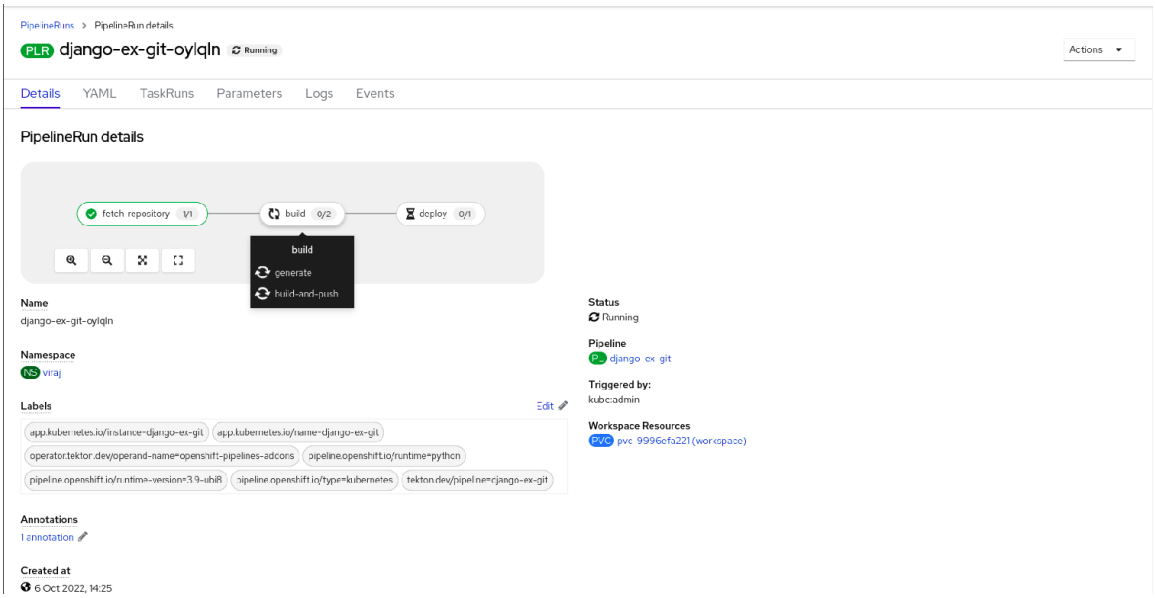
5. **PipelineRun details** ページには、実行されるパイプラインが表示されます。パイプラインが開始すると、タスクおよび各タスク内のステップが実行します。以下を行うことができます。

- **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。

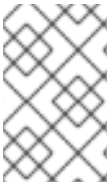
- タスクにカーソルを合わせると、各ステップの実行にかかった時間、タスク名、タスクステータスなどの詳細が表示されます。
- タスクバッジにカーソルを合わせ、完了したタスクとタスクの合計数を確認します。
- タスクをクリックし、タスクの各ステップのログを表示します。
- **Logs** タブをクリックして、タスクの実行シーケンスに関連するログを表示します。該当するボタンを使用して、ペインをデプロイメントし、ログを個別に、または一括してダウンロードすることもできます。
- **Events** タブをクリックして、パイプライン実行で生成されるイベントのストリームを表示します。
  Task Runs、**Logs**、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



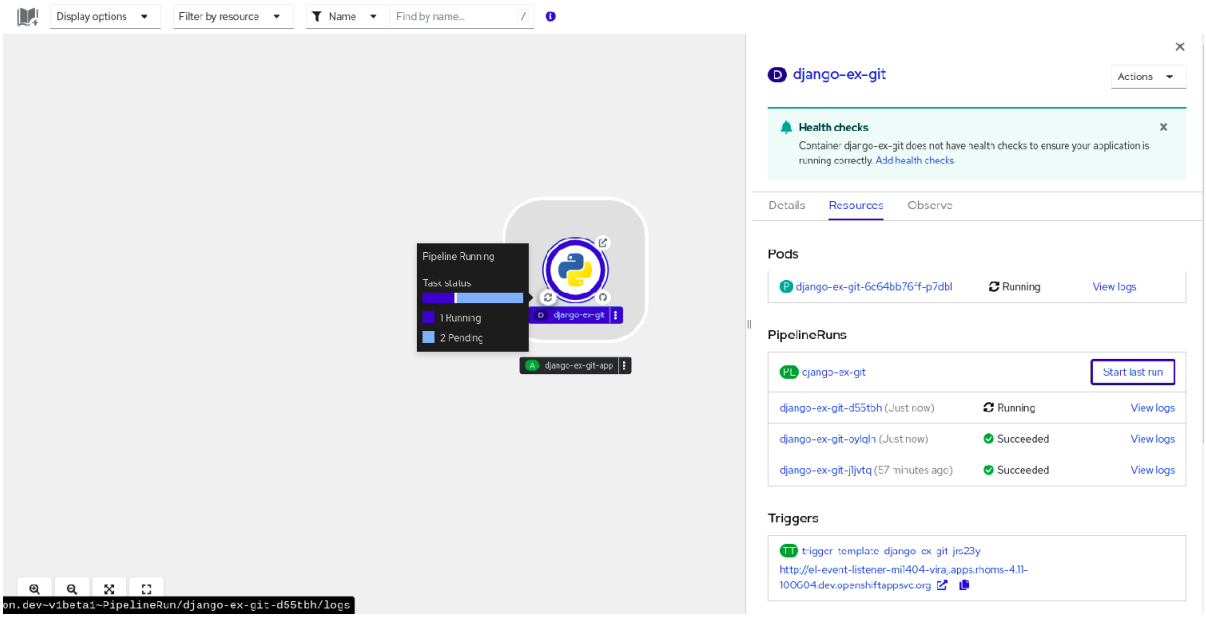
注記

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は、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、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ことがあります。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。

重要

OpenShift Pipelines コンソールプラグインには、OpenShift Container Platform バージョン 4.15 以降が必要です。

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。
- 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。
- 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、タスク、およびステップアクションの指定

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。

StepAction カスタムリソース (CR) で定義されるステップアクションは、タスク内のステップ 1 つで完了する再利用可能なアクションです。ステップを指定する場合は、既存のソースから **StepAction** 定義を参照できます。

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、タスク、または **StepAction** 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、パイプライン、または **StepAction** 定義を取得します。

バンドルリゾルバー

OpenShift コンテナーリポジトリーなどの任意の OCI リポジトリーから入手できる OCI イメージである Tekton バンドルからタスク、パイプライン、または **StepAction** 定義を取得します。

Git リゾルバー

Git リポジトリーからタスク、パイプライン、または **StepAction** 定義を取得します。リポジトリー、ブランチ、パスを指定する必要があります。

HTTP リゾルバー

リモート HTTP または HTTPS の URL からタスク、パイプライン、または **StepAction** 定義を取得します。認証の URL を指定する必要があります。

クラスターリゾルバー

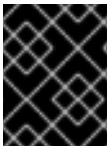
特定の namespace の同じ OpenShift Container Platform クラスター上にすでに作成されているタスク、パイプライン、または **StepAction** 定義を取得します。

OpenShift Pipelines のインストールには、パイプラインで使用できる一連の標準タスクが含まれています。これらのタスクは、OpenShift Pipelines インストール namespace (通常は **openshift-pipelines** namespace) にあります。クラスターリゾルバーを使用してタスクにアクセスできます。

OpenShift Pipelines は標準の **StepAction** 定義も提供します。クラスターリゾルバーを使用して、この定義にアクセスできます。

3.1. TEKTON カタログからのリモートパイプライン、タスク、またはステップアクションの指定

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るリモートパイプライン、タスク、または **StepAction** 定義を指定できます。



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** です。

(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、タスク、またはステップアクションの指定

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。タスク内でステップを作成するときに、Artifact Hub または Tekton Hub からリモート **StepAction** 定義を参照できます。

手順

- Artifact Hub または Tekton Hub からリモートパイプライン、タスク、または **StepAction** 定義を指定するには、**pipelineRef**、**taskRef**、または **step.ref** 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	説明	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**

パラメーター	説明	値の例
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、パイプライン、パイプライン実行、またはタスク実行の仕様の **params** セクションでこれらのパラメーターの値を指定します。**pipelineRef** または **taskRef** 仕様の **params** セクションには、リゾルバーがサポートするパラメーターのみを含める必要があります。

例

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"
  params:
  - 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-hub-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"
    params:
    - 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: hub-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"
  params:
  - name: sample-task-parameter
    value: test
```

次のタスクの例には、カタログから **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: hub-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
    - resolver: hub
    - params:
      - name: catalog
        value: tekton-catalog-stepactions
      - name: type
        value: artifact
      - name: kind
```

```
    value: StepAction
  - name: name
    value: example-stepaction
  - name: version
    value: "0.6"
params:
- name: sample-stepaction-parameter
  value: test
```

3.2. TEKTON バンドルからのリモートパイプライン、タスク、またはステップアクションの指定

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、タスク、または **StepAction** 定義を指定できます。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、タスク、またはステップアクションの指定

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。タスク内でステップを作成する場合、Tekton バンドルからリモート **StepAction** 定義を参照できます。

手順

- Tekton バンドルからリモートパイプライン、タスク、または **StepAction** 定義を指定するには、**pipelineRef**、**taskRef**、または **step.ref** 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	説明	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、パイプライン、パイプライン実行、またはタスク実行の仕様の **params** セクションでこれらのパラメーターの値を指定します。**pipelineRef** または **taskRef** 仕様の **params** セクションには、リゾルバーがサポートするパラメーターのみを含める必要があります。

例

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
  params:
  - name: sample-pipeline-parameter
    value: test
  - 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
    params:
    - 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
  params:
  - name: sample-task-parameter
    value: test
```

次のタスクの例には、Tekton バンドルから **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
```

```
  name: bundle-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
      resolver: bundles
      params:
      - name: bundle
        value: registry.example.com:5000/simple/new_task:latest
      - name: name
        value: hello-world-action
      - name: kind
        value: StepAction
    params:
    - name: sample-stepaction-parameter
      value: test
```

3.3. 匿名 GIT クローンでのリモートパイプライン、タスク、またはステップアクションの指定

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、タスク、または **StepAction** 定義にアクセスできます。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。匿名アクセスの場合、認証情報を必要とせずにリゾルバーを使用してリポジトリーを複製できます。

3.3.1. 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は、匿名クローン作成用のデフォルトの Git リポジトリー URL。

3.3.2. 匿名クローン作成に Git リゾルバーを使用したリモートパイプライン、タスク、またはステップアクションの指定

パイプライン実行を作成するときに、匿名クローンを使用して 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。タスク内でステップを作成するときに、Git リポジトリーからリモート **StepAction** 定義を参照できます。

手順

- Git リポジトリーからリモートパイプライン、タスク、または **StepAction** 定義を指定するには、**pipelineRef**、**taskRef**、または **step.ref** 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3.3 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	説明	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**
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。**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。

例

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/v1beta1
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
  params:
    - name: name
      value: "testPipelineRun"
    - name: sample-pipeline-parameter
      value: test
```

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
      params:
      - 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/v1beta1
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
  params:
  - name: sample-task-parameter
    value: test
```

以下のタスク例には、Git リポジトリーから **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: git-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
      resolver: git
      - name: url
        value: https://github.com/openshift-pipelines/tektoncd-catalog.git
      - name: revision
        value: p
      - name: pathInRepo
        value: stepactions/stepaction-git-clone/0.4.1/stepaction-git-clone.yaml
    params:
    - name: sample-stepaction-parameter
      value: test
```

3.4. 認証された GIT API でのリモートパイプライン、タスク、またはステップアクションの指定

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、タスク、または **StepAction** 定義を指定できます。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。ユーザー認証をサポートする認証済み API を使用すると、リポジトリーに安全にアクセスできます。

3.4.1. 認証された API の Git リゾルバーの設定

認証されたソースコントロール管理 (SCM) API の場合は、認証された Git 接続の設定を指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 データセンター
- Bitbucket Cloud

注記

- 認証された SCM API を使用して Git 接続を設定できます。クラスター上のすべてのユーザーが 1 つのリポジトリーにアクセスできるようにセキュリティートークンを提供できます。さらに、特定のパイプラインまたはタスクに対して異なる SCM プロバイダーとトークンを指定することもできます。
- 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。

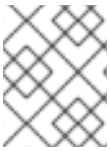
(4) 認証された SCM API で使用するベース URL。**github.com**、**gitlab.com**、または Bitbucket Cloud を使用している場合、この設定は必要ありません。

(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 namespace (**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。

3.4.2. 複数の Git プロバイダーの設定

複数の Git プロバイダーを設定するか、同じ Git プロバイダーに複数の設定を追加して、異なるタスク実行とパイプライン実行で使用できます。

一意の識別子鍵の接頭辞を使用して、**TektonConfig** カスタムリソース (CR) に詳細を追加します。

手順

1. 次のコマンドを実行して、**TektonConfig** CR を編集します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** CR 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:
      # configuration 1 (1)
      fetch-timeout: "1m"
      default-url: "https://github.com/tektoncd/catalog.git"
      default-revision: "main"
      scm-type: "github"
      server-url: ""
      api-token-secret-name: ""
      api-token-secret-key: ""
      api-token-secret-namespace: "default"
      default-org: ""
      # configuration 2 (2)
      test1.fetch-timeout: "5m"
      test1.default-url: ""
      test1.default-revision: "stable"
      test1.scm-type: "github"
      test1.server-url: "api.internal-github.com"
      test1.api-token-secret-name: "test1-secret"
      test1.api-token-secret-key: "token"
      test1.api-token-secret-namespace: "test1"
```

```
    test1.default-org: "tektoncd"
    # configuration 3 (3)
    test2.fetch-timeout: "10m"
    test2.default-url: ""
    test2.default-revision: "stable"
    test2.scm-type: "gitlab"
    test2.server-url: "api.internal-gitlab.com"
    test2.api-token-secret-name: "test2-secret"
    test2.api-token-secret-key: "pat"
    test2.api-token-secret-namespace: "test2"
    test2.default-org: "tektoncd-infra"
# ...
```

1. **configKey** 鍵が提供されない場合、またはキーが **default** 値で提供されている場合に使用するデフォルトの設定。
2. **configKey** 鍵が **test1** 値で渡される場合に使用される設定。
3. **configKey** 鍵が **test2** 値で渡される場合に使用される設定。

警告

. 記号が付いた **configKey** 値はサポートされません。**.** シンボルを含む **configKey** 値を渡そうとすると、値を渡した **TaskRun** または **PipelineRun** リソースの実行に失敗します。

3.4.3. 認証された SCM API で Git リゾルバーを使用したリモートパイプライン、タスク、またはステップアクションの指定

パイプライン実行を作成するときに、認証された SCM API を使用して 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。タスク内でステップを作成するときに、Git リポジトリーからリモート **StepAction** 定義を参照できます。

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

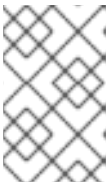
- Git リポジトリーからリモートパイプライン、タスク、または **StepAction** 定義を指定するには、**pipelineRef**、**taskRef**、または **step.ref** 仕様で次の参照形式を使用します。

```
# ...
  resolver: git
  params:
  - name: org
    value: <git_organization_name>
```

```
    - name: repo
      value: <git_repository_name>
    - 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	説明	値の例
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**
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**
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、パイプライン、パイプライン実行、またはタスク実行の仕様の **params** セクションでこれらのパラメーターの値を指定します。**pipelineRef** または **taskRef** 仕様の **params** セクションには、リゾルバーがサポートするパラメーターのみを含める必要があります。

例

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: org
      value: tektoncd
```

```
    - name: repo
      value: catalog
    - name: revision
      value: main
    - name: pathInRepo
      value: pipeline/simple/0.1/simple.yaml
  params:
  - name: name
    value: "testPipelineRun"
  - name: sample-pipeline-parameter
    value: test
```

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: org
        value: tektoncd
      - name: repo
        value: catalog
      - name: revision
        value: main
      - name: pathInRepo
        value: task/git-clone/0.6/git-clone.yaml
    params:
    - 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: org
      value: tektoncd
    - name: repo
      value: catalog
    - name: revision
      value: main
    - name: pathInRepo
      value: task/git-clone/0.6/git-clone.yaml
```

```
  params:
  - name: sample-task-parameter
    value: test
```

以下のタスク例には、Git リポジトリーから **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: git-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
      resolver: git
      - name: org
        value: openshift-pipelines
      - name: repo
        value: tektoncd-catalog
      - name: revision
        value: p
      - name: pathInRepo
        value: stepactions/stepaction-git-clone/0.4.1/stepaction-git-clone.yaml
    params:
    - name: sample-stepaction-parameter
      value: test
```

3.4.4. 複数の Git プロバイダーの指定

TaskRun および **PipelineRun** リソースを作成するときに、一意の **configKey** パラメーターを渡すことで、複数の Git プロバイダーを指定できます。

configKey パラメーターが渡されない場合、デフォルト設定が使用されます。**configKey** の値を **default** に設定して、デフォルト設定を指定することもできます。

警告

. 記号が付いた **configKey** 値はサポートされません。**.** シンボルを含む **configKey** 値を渡そうとすると、値を渡した **TaskRun** または **PipelineRun** リソースの実行に失敗します。

前提条件

- **Tektonconfig** カスタムリソースを使用して複数の Git プロバイダーを設定します。詳細は、「複数の Git プロバイダーの設定」を参照してください。

手順

- Git プロバイダーを指定するには、**pipelineRef** および **taskRef** 仕様で以下の参照形式を使用します。

```
# ...
  resolver: git
  params:
  # ...
  - name: configKey
    value: <your_unique_key> 1
# ...
```

1 設定キーのいずれかに一致する一意のキー (例: **test1**）。

3.4.5. Git リゾルバーの設定を上書きする認証済み SCM API を使用した Git リゾルバーでのリモートパイプラインまたはタスクの指定

特定のパイプライン実行またはタスクの初期設定をオーバーライドして、さまざまなユースケースに応じて動作をカスタマイズできます。この方法を使用して、**TektonConfig** カスタムリソース (CR) で設定されていない認証済みプロバイダーにアクセスできます。

次のタスク実行例では、以前のリゾルバー設定をオーバーライドする Git リポジトリーからのリモートタスクを参照します。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: git-task-reference-demo
spec:
  taskRef:
    resolver: git
    params:
    - name: org
      value: tektoncd
    - name: repo
      value: catalog
    - name: revision
      value: main
    - name: pathInRepo
      value: task/git-clone/0.6/git-clone.yaml
    - name: token
      value: my-secret-token
    - name: tokenKey
      value: token
    - name: scmType
      value: github
    - name: serverURL
      value: https://ghe.mycompany.com
```

表3.5 Git リゾルバーをオーバーライドするためのサポート対象のパラメーター

パラメーター	説明	値の例
org	リポジトリーの組織。	**tektoncd**
repo	リポジトリー名。	catalog

パラメーター	説明	値の例
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。	main
pathInRepo	リポジトリー内の YAML ファイルのパス名。	task/git-clone/0.6/git-clone.yaml
トークン (token)	認証に使用されるシークレット名。	my-secret-token
tokenKey	トークンのキー名。	トークン (token)
scmType	SCM (ソースコントロール管理) システムのタイプ。	github
serverURL	リポジトリーをホストしているサーバーの URL。	**https://ghe.mycompany.com**

3.5. HTTP リゾルバーを使用したリモートパイプライン、タスク、またはステップアクションの指定

HTTP リゾルバーを使用して、HTTP または HTTPS URL からリモートパイプライン、タスク、または **StepAction** 定義を指定できます。URL は、パイプライン、タスク、またはステップアクションを定義する YAML ファイルを参照する必要があります。

3.5.1. HTTP リゾルバーの設定

HTTP リゾルバーを使用して、HTTP または HTTPS URL からパイプラインまたはタスクを取得できます。**TektonConfig** カスタムリソース (CR) を編集して、HTTP リゾルバーのデフォルト値を設定できます。

手順

1. 次のコマンドを入力して、**TektonConfig** CR を編集します。

```
$ oc edit TektonConfig config
```

2. **TektonConfig** CR で、**pipeline.http-resolver-config** 仕様を編集します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    http-resolver-config:
      fetch-timeout: "1m" (1)
```

1 HTTP リゾルバーがサーバーからの応答を待機する最大時間。

3.5.2. HTTP Resolver でのリモートパイプライン、タスク、またはステップアクションの指定

パイプライン実行を作成するときに、HTTP または HTTPS URL 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、HTTP または HTTPS URL からリモートタスクを指定できます。タスク内でステップを作成する場合、HTTP または HTTPS URL からリモート **StepAction** 定義を参照できます。

手順

- **pipelineRef**、**taskRef**、または **step.ref** 仕様で次の形式を使用して、HTTP または HTTPS URL からリモートパイプライン、タスク、または **StepAction** 定義を指定します。

```
# ...
  resolver: http
  params:
  - name: url
    value: <fully_qualified_http_url>
# ...
```

表3.6 HTTP 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	説明	値の例
url	取得する Tekton リソースを指す HTTP URL。	**https://raw.githubusercontent.com/openshift-pipelines/tektoncd-catalog/p/tasks/task-git-clone/0.4.1/task-git-clone.yaml**

例

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/v1beta1
kind: PipelineRun
metadata:
  name: http-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: http
    params:
    - name: url
      value: https://raw.githubusercontent.com/tektoncd/catalog/main/pipeline/build-push-gke-deploy/0.1/build-push-gke-deploy.yaml
  params:
  - name: sample-pipeline-parameter
    value: test
  - name: username
    value: "pipelines"
```

次のパイプラインの例では、HTTPS URL からリモートタスクを参照するタスクを定義します。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-with-http-task-reference-demo
spec:
  tasks:
  - name: "http-task-demo"
    taskRef:
      resolver: http
      params:
      - name: url
        value: https://raw.githubusercontent.com/openshift-pipelines/tektoncd-catalog/p/tasks/task-git-clone/0.4.1/task-git-clone.yaml
    params:
    - name: sample-task-parameter
      value: test
```

次のタスク実行例では、HTTPS URL からリモートタスクを参照します。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: http-task-reference-demo
spec:
  taskRef:
    resolver: http
    params:
    - name: url
      value: https://raw.githubusercontent.com/openshift-pipelines/tektoncd-catalog/p/tasks/task-git-clone/0.4.1/task-git-clone.yaml
  params:
  - name: sample-task-parameter
    value: test
```

次のタスクの例には、HTTPS URL から **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: http-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
      resolver: http
      params:
      - name: url
        value: https://raw.githubusercontent.com/openshift-pipelines/tektoncd-catalog/p/stepactions/stepaction-git-clone/0.4.1/stepaction-git-clone.yaml
    params:
    - name: sample-stepaction-parameter
      value: test
```

3.6. 同じクラスターからのパイプライン、タスク、またはステップアクションの指定

クラスターリゾルバーを使用して、Red Hat OpenShift Pipelines が実行されている OpenShift Container Platform クラスターの namespace で定義されるパイプライン、タスク、または **StepAction** 定義を指定できます。

特に、クラスターリゾルバーを使用して、OpenShift Pipelines がインストール namespace (通常は **openshift-pipelines** namespace) で提供するタスクにアクセスできます。

3.6.1. 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 namespace を変更したり、クラスターリゾルバーが使用できる namespace 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は、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) リゾルバーのアクセスがブロックされる 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。

3.6.2. クラスターリゾルバーを使用した同じクラスターからのパイプライン、タスク、またはステップアクションの指定

パイプライン実行を作成するときに、同じクラスター上に存在する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、同じクラスター上に存在するタスクを指定できます。タスク内でステップを作成する場合は、同じクラスターに存在する **StepAction** 定義を指定できます。

手順

- 同じクラスターからパイプライン、タスク、または **StepAction** 定義を指定するには、**pipelineRef**、**taskRef**、または **step.ref** 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|stepaction]
# ...
```

表3.7 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	説明	値の例
name	取得するリソースの名前。	**some-pipeline**
namespace	リソースを含むクラスター内の namespace。	**other-namespace**
kind	取得するリソースの種類。	**pipeline**

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。

例

次のパイプライン実行の例は、同じクラスターからの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
  params:
  - 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
    params:
    - name: sample-task-parameter
      value: test
```

次のタスク実行例では、同じクラスターの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
  params:
  - name: sample-task-parameter
    value: test
```

次のタスクの例には、同じクラスターから **StepAction** 定義を参照するステップが含まれています。

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: cluster-stepaction-reference-demo
spec:
  steps:
  - name: example-step
    ref:
      resolver: cluster
      params:
      - name: name
        value: some-step
      - name: namespace
```

```
      value: test-namespace
    - name: kind
      value: stepaction
  params:
  - name: sample-stepaction-parameter
    value: test
```

3.7. OPENSHIFT PIPELINES NAMESPACE で提供されるタスク

OpenShift Pipelines のインストールには、パイプラインで使用できる一連の標準タスクが含まれています。これらのタスクは、OpenShift Pipelines インストール namespace (通常は **openshift-pipelines** namespace) にあります。クラスターリゾルバーを使用してタスクにアクセスできます。

バージョン 1.16 まで、OpenShift Pipelines には **ClusterTask** 機能が含まれていました。バージョン 1.17 以降にはこの機能が含まれなくなりました。パイプラインが **ClusterTask** 参照を使用する場合、クラスターリゾルバーを使用して、OpenShift Pipelines インストール namespace で利用可能なタスクでそれらを再作成できます。ただし、既存の **ClusterTask** 定義と比較して、これらのタスクに特定の変更が加えられています。

OpenShift Pipelines インストール namespace で利用可能なタスクのいずれかでカスタム実行イメージを指定することはできません。これらのタスクは、**BUILDER_IMAGE**、**gitInitImage**、**KN_IMAGE** などのパラメーターをサポートしません。カスタム実行イメージを使用する場合は、タスクのコピーを作成し、そのコピーを編集してイメージを置き換えます。

buildah

buildah タスクは、ソースコードツリーをコンテナーイメージにビルドし、そのイメージをコンテナーレジストリーにプッシュします。

buildah タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: buildah
      - name: namespace
        value: openshift-pipelines
    params:
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.8 buildah タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	Buildah によってビルドされる完全修飾コンテナーイメージ名。	**string**	
DOCKERFILE	**source** ワークスペースを基準とした **Dockerfile** (または **Containerfile**) へのパス。	**string**	**./Dockerfile**
CONTEXT	コンテキストとして使用するディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**

表3.9 buildah タスクでサポートされているワークスペース

ワークスペース	説明
source	コンテナービルドコンテキスト。通常、**Dockerfile** または **Containerfile** ファイルが含まれるアプリケーションのソースコードです。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

ワークスペース	説明
rhel-entitlement	Buildah が Red Hat Enterprise Linux (RHEL) サブスクリプションへのアクセスに使用するエンタイトルメントキーを提供するためのオプションのワークスペース。マウントされたワークスペースには、**entitlement.pem** ファイルと **entitlement-key.pem** ファイルが含まれている必要があります。

表3.10 buildah タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

buildah ClusterTaskからの変更点

- **VERBOSE** パラメーターが追加されました。
- **BUILDER_IMAGE** パラメーターが削除されました。

git-cli

git-cli タスクは、**git** コマンドラインユーティリティーを実行します。**GIT_SCRIPT** パラメーターを使用して、完全な Git コマンドまたは複数のコマンドを渡すことができます。たとえば、プッシュを完了するためにコマンドが Git リポジトリーへの認証を必要とする場合は、認証のクレデンシャルを指定する必要があります。

git-cli タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: update-repo
spec:
# ...
  tasks:
# ...
  - name: push-to-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-cli
      - name: namespace
        value: openshift-pipelines
    params:
    - name: GIT_SCRIPT
```

```
      value: "git push"
    - name: GIT_USER_NAME
      value: "Example Developer"
    - name: GIT_USER_EMAIL
      value: "developer@example.com"
    workspaces:
    - name: ssh-directory
      workspace: ssh-workspace (1)
    - name: source
      workspace: shared-workspace
# ...
```

(1) この例では、**ssh-workspace** には、Git リポジトリーへの認可に有効なキーを含む **.ssh** ディレクトリーの内容が含まれている必要があります。

表3.11 git-cli タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
CRT_FILENAME	**ssl-ca-directory** ワークスペース内の認証局 (CA) のバンドルファイル名。	**string**	**ca-bundle.crt**
HTTP_PROXY	HTTP プロキシーサーバー (TLS 以外のリクエスト)。	**string**	
HTTPS_PROXY	HTTPS プロキシーサーバー (TLS リクエスト)。	**string**	
NO_PROXY	HTTP/HTTPS リクエストのプロキシーをオプトアウトします。	**string**	
SUBDIRECTORY	Git リポジトリーが存在する **source** ワークスペースへの相対パス。	**string**	
USER_HOME	Pod 内の Git ユーザーのホームディレクトリーへの絶対パス。	**string**	**/home/git**
DELETE_EXISTING	Git 操作を完了する前に、**source** ワークスペースの既存のコンテンツをすべて消去します。	**string**	**true**
VERBOSE	実行したすべてのコマンドをログに記録します。	**string**	**false**
SSL_VERIFY	グローバル **http.sslVerify** 値。リモートリポジトリーを信頼していない限り、**false** を使用しないでください。	**string**	**true**
GIT_USER_NAME	Git 操作を実行するための Git ユーザー名。	**string**	

パラメーター	説明	型	デフォルト値
GIT_USER_EMAIL	Git 操作を実行するための Git ユーザーのメール。	**string**	
GIT_SCRIPT	実行する Git スクリプト。	**string**	**git help**

表3.12 git-cli タスクでサポートされているワークスペース

ワークスペース	説明
ssh-directory	必要に応じて、秘密鍵、**known_hosts**、**config** などのファイルを含む **.ssh** ディレクトリー。このワークスペースを指定すると、タスクは Git リポジトリーへの認証にこのワークスペースを使用します。認証情報を安全に保存するために、このワークスペースを **Secret** リソースにバインドします。
basic-auth	**.gitconfig** および **.git-credentials** ファイルが含まれるワークスペース。このワークスペースを指定すると、タスクは Git リポジトリーへの認証にこのワークスペースを使用します。可能な場合は、認証に **basic-auth** ではなく **ssh-directory** ワークスペースを使用します。認証情報を安全に保存するために、このワークスペースを **Secret** リソースにバインドします。
ssl-ca-directory	CA 証明書を含むワークスペース。このワークスペースを指定すると、Git は HTTPS を使用してリモートリポジトリーとやり取りする場合に、これらの証明書を使用してピアを検証します。
source	取得した Git リポジトリーを含むワークスペース。
input	Git リポジトリーに追加する必要があるファイルが含まれるオプションのワークスペース。**$(workspaces.input.path)** を使用して、スクリプトからワークスペースにアクセスできます。以下に例を示します。 **cp $(workspaces.input.path)/<file_that_i_want> .** **git add <file_that_i_want>**

表3.13 git-cli タスクが返す結果

結果	型	説明
COMMIT	**string**	クローンされた Git リポジトリー内の現在のブランチの HEAD にあるコミットの SHA ダイジェスト。

git-cli ClusterTask からの変更点

- 新しいパラメーターが複数追加されました。
- **BASE_IMAGE** パラメーターが削除されました。
- **ssl-ca-directory** ワークスペースが追加されました。

- **USER_HOME** および **VERBOSE** パラメーターのデフォルト値が変更されました。
- 結果の名前が **commit** から **COMMIT** に変更されました。

git-clone

git-clone タスクは、Git を使用してワークスペース上のリモートリポジトリーを初期化し、クローンを作成します。このタスクは、このソースコードをビルドまたは処理するパイプラインの開始時に使用できます。

git-clone タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-source
spec:
# ...
  tasks:
  - name: clone-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    params:
    - name: URL
      value: "https://github.com/example/repo.git"
    workspaces:
    - name: output
      workspace: shared-workspace
```

表3.14 git-clone タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
CRT_FILENAME	**ssl-ca-directory** ワークスペース内の認証局 (CA) のバンドルファイル名。	**string**	**ca-bundle.crt**
HTTP_PROXY	HTTP プロキシーサーバー (TLS 以外のリクエスト)。	**string**	
HTTPS_PROXY	HTTPS プロキシーサーバー (TLS リクエスト)。	**string**	
NO_PROXY	HTTP/HTTPS リクエストのプロキシーをオプトアウトします。	**string**	
SUBDIRECTORY	タスクが Git リポジトリーを配置する **output** ワークスペース内の相対パス。	**string**	

パラメーター	説明	型	デフォルト値
USER_HOME	Pod 内の Git ユーザーのホームディレクトリーへの絶対パス。	**string**	**/home/git**
DELETE_EXISTING	Git 操作を実行する前に、デフォルトのワークスペースにコンテンツが存在する場合は削除します。	**string**	**true**
VERBOSE	実行したコマンドをログに記録します。	**string**	**false**
SSL_VERIFY	グローバル **http.sslVerify** 値。リモートリポジトリーを信頼していない限り、このパラメーターを **false** に設定しないでください。	**string**	**true**
URL	Git リポジトリー URL	**string**	
リビジョン	チェックアウトするリビジョン (ブランチやタグなど)。	**string**	**main**
REFSPEC	リビジョンをチェックアウトする前にタスクが取得するリポジトリーの **refspec** 文字列。	**string**	
SUBMODULES	Git サブモジュールを初期化して取得します。	**string**	**true**
DEPTH	取得するコミットの数。"シャロークローン" は単一のコミットです。	**string**	**1**
SPARSE_CHECKOUT_DIRECTORIES	"スパースチェックアウト" を実行するための、コンマで区切られたディレクトリーパターンのリスト。	**string**	

表3.15 git-clone タスクでサポートされているワークスペース

ワークスペース	説明
ssh-directory	必要に応じて、秘密鍵、**known_hosts**、**config** などのファイルを含む **.ssh** ディレクトリー。このワークスペースを指定すると、タスクは Git リポジトリーへの認証にこのワークスペースを使用します。認証情報を安全に保存するために、このワークスペースを **Secret** リソースにバインドします。
basic-auth	**.gitconfig** および **.git-credentials** ファイルが含まれるワークスペース。このワークスペースを指定すると、タスクは Git リポジトリーへの認証にこのワークスペースを使用します。可能な場合は、認証に **basic-auth** ではなく **ssh-directory** ワークスペースを使用します。認証情報を安全に保存するために、このワークスペースを **Secret** リソースにバインドします。

ワークスペース	説明
ssl-ca-directory	CA 証明書を含むワークスペース。このワークスペースを指定すると、Git は HTTPS を使用してリモートリポジトリーとやり取りする場合に、これらの証明書を使用してピアを検証します。
出力 (output)	取得した Git リポジトリーを含むワークスペース。データはワークスペースのルートまたは **SUBDIRECTORY** パラメーターで定義された相対パスに配置されます。

表3.16 git-clone タスクが返す結果

結果	型	説明
COMMIT	**string**	クローンされた Git リポジトリー内の現在のブランチの HEAD にあるコミットの SHA ダイジェスト。
URL	**string**	クローンされたリポジトリーの URL。
COMMITTER_DATE	**string**	クローンされた Git リポジトリー内の現在のブランチの HEAD にあるコミットのエポックタイムスタンプ。

git-clone ClusterTask からの変更点

- すべてのパラメーター名が大文字に変更されました。
- すべての結果名が大文字に変更されました。
- **gitInitImage** パラメーターが削除されました。

kn
kn タスクは、**kn** コマンドラインユーティリティーを使用して、サービス、リビジョン、ルートなどの Knative リソースでの操作を完了します。

kn タスクの使用例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: kn-run
spec:
 pipelineSpec:
   tasks:
   - name: kn-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
```

```
      value: kn
    - name: namespace
      value: openshift-pipelines
  params:
  - name: ARGS
    value: [version]
```

表3.17 kn タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
ARGS	**kn** ユーティリティーの引数。	**array**	**- help**

kn ClusterTask からの変更点

- **KN_IMAGE** パラメーターが削除されました。

kn-apply

kn-apply タスクは、指定されたイメージを Knative サービスにデプロイします。このタスクは、 **kn service apply** コマンドを使用して、指定された Knative サービスを作成または更新します。

kn-apply タスクの使用例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: kn-apply-run
spec:
 pipelineSpec:
   tasks:
   - name: kn-apply-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: kn-apply
       - name: namespace
         value: openshift-pipelines
     params:
     - name: SERVICE
       value: "hello"
     - name: IMAGE
       value: "gcr.io/knative-samples/helloworld-go:latest"
```

表3.18 kn-apply タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
サービス	Knative サービス名。	**string**	

パラメーター	説明	型	デフォルト値
IMAGE	デプロイするイメージの完全修飾名。	**string**	

kn-apply ClusterTask からの変更点

- **KN_IMAGE** パラメーターが削除されました。

maven
Maven タスクは Maven ビルドを実行します。

Maven タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-from-source
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: maven
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.19 Maven タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
GOALS	実行する Maven の目標。	**array**	**- package**
MAVEN_MIRROR_URL	Maven リポジトリーのミラー URL。	**string**	
SUBDIRECTORY	タスクが Maven ビルドを実行する **source** ワークスペース内のサブディレクトリー。	**string**	**.**

表3.20 Maven タスクでサポートされているワークスペース

ワークスペース	説明
source	Maven プロジェクトが含まれるワークスペース。
server_secret	ユーザー名やパスワードなど、Maven サーバーに接続するためのシークレットが含まれるワークスペース。
proxy_secret	ユーザー名やパスワードなど、プロキシーサーバーに接続するための認証情報が含まれるワークスペース。
proxy_configmap	**proxy_port**、**proxy_host**、**proxy_protocol**、**proxy_non_proxy_hosts** などのプロキシー設定値が含まれるワークスペース。
maven_settings	カスタム Maven 設定が含まれるワークスペース。

Maven ClusterTask からの変更

- **CONTEXT_DIR** のパラメーター名が **SUBDIRECTORY** に変更されました。
- **maven-settings** のワークスペース名が **maven_settings** に変更されました。

openshift-client

openshift-client タスクは、**oc** コマンドラインインターフェイスを使用してコマンドを実行します。このタスクを使用して、OpenShift Container Platform クラスターを管理できます。

openshift-client タスクの使用例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: openshift-client-run
spec:
 pipelineSpec:
   tasks:
   - name: openshift-client-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: openshift-client
       - name: namespace
         value: openshift-pipelines
     params:
     - name: SCRIPT
       value: "oc version"
```

表3.21 openshift-client タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
SCRIPT	実行する **oc** CLI 引数。	**string**	**oc help**
VERSION	使用する OpenShift Container Platform バージョン。	**string**	**latest**

表3.22 **openshift-client タスクでサポートされているワークスペース**

ワークスペース	説明
manifest_dir	**oc** ユーティリティーを使用して適用するマニフェストファイルが含まれるワークスペース。
kubeconfig_dir	クラスターにアクセスするための認証情報を含む **.kube/config** ファイルを提供できるオプションのワークスペース。このファイルをワークスペースのルートに配置し、**kubeconfig** という名前を付けます。

openshift-client ClusterTask からの変更点

- ワークスペース名 **manifest-dir** が **manifest_dir** に変更されました。
- ワークスペース名 **kubeconfig-dir** が **kubeconfig_dir** に変更されました。

s2i-dotnet

s2i-dotnet タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/dotnet** として入手できる Source to Image (S2I) dotnet ビルダーイメージを使用してソースコードをビルドします。

s2i-dotnet タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-dotnet
      - name: namespace
        value: openshift-pipelines
    workspaces:
```

```
    - name: source
      workspace: shared-workspace
  # ...
```

表3.23 s2i-dotnet タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.24 s2i-dotnet タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.25 s2i-dotnet タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-go

s2i-go タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/golang** として入手できる S2I Golang ビルダーイメージを使用してソースコードをビルドします。

s2i-go タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-go
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.26 s2i-go タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.27 s2i-go タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。

ワークスペース	説明
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.28 s2i-go タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-java
s2i-java タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/java** として入手できる S2I Java ビルダーイメージを使用してソースコードをビルドします。

表3.29 s2i-java タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**

パラメーター	説明	型	デフォルト値
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.30 s2i-java タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.31 s2i-java タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-java ClusterTask からの変更点

- 新しいパラメーターが複数追加されました。
- **BUILDER_IMAGE**、**MAVEN_ARGS_APPEND**、**MAVEN_CLEAR_REPO**、および **MAVEN_MIRROR_URL** パラメーターが削除されました。**MAVEN_ARGS_APPEND**、**MAVEN_CLEAR_REPO** および **MAVEN_MIRROR_URL** の値を環境変数として渡すことができます。
- パラメーター名 **PATH_CONTEXT** が **CONTEXT** に変更されました。

- パラメーター名 **TLS_VERIFY** が **TLSVERIFY** に変更されました。
- **IMAGE_URL** の結果が追加されました。

s2i-nodejs

s2i-nodejs タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/nodejs** として入手できる S2I NodeJS ビルダーイメージを使用してソースコードをビルドします。

s2i-nodejs タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-nodejs
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.32 s2i-nodejs タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**

パラメーター	説明	型	デフォルト値
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.33 s2i-nodejs タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.34 s2i-nodejs タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-perl

s2i-perl タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/perl** として入手できる S2I Perl ビルダーイメージを使用してソースコードをビルドします。

s2i-perl タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-perl
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.35 **s2i-perl タスクでサポートされているパラメーター**

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**

パラメーター	説明	型	デフォルト値
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.36 s2i-perl タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.37 s2i-perl タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-php

s2i-php タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/php** として入手できる S2I PHP ビルダーイメージを使用してソースコードをビルドします。

s2i-php タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-php
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.38 s2i-php タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	

パラメーター	説明	型	デフォルト値
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.39 s2i-php タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.40 s2i-php タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-python

s2i-python タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/Python** として入手できる S2I Python ビルダーイメージを使用してソースコードをビルドします。

s2i-python タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
```

```
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-python
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.41 s2i-python タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**

パラメーター	説明	型	デフォルト値
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.42 s2i-python タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.43 s2i-python タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

s2i-ruby

s2i-ruby タスクは、OpenShift Container Platform レジストリーから **image-registry.openshift-image-registry.svc:5000/openshift/ruby** として入手できる S2I Ruby ビルダーイメージを使用してソースコードをビルドします。

s2i-ruby タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
```

```
      - name: kind
        value: task
      - name: name
        value: s2i-ruby
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
# ...
```

表3.44 s2i-ruby タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
IMAGE	S2I プロセスがビルドするコンテナーイメージの完全修飾名。	**string**	
IMAGE_SCRIPTS_URL	ビルダーイメージのデフォルトのアセンブルおよび実行スクリプトを含む URL。	**string**	**image:///usr/libexec/s2i**
ENV_VARS	ビルドプロセスで設定する環境変数の値の配列。**KEY=VALUE** 形式でリストされます。	**array**	
CONTEXT	コンテキストとして使用する **source** ワークスペース内のディレクトリーへのパス。	**string**	**.**
STORAGE_DRIVER	Buildah ストレージドライバーを設定して、現在のクラスターノード設定を反映します。	**string**	**vfs**
FORMAT	ビルドするコンテナーの形式。**oci** または **docker** のいずれかです。	**string**	**oci**
BUILD_EXTRA_ARGS	イメージをビルドするときの **build** コマンドの追加パラメーター。	**string**	
PUSH_EXTRA_ARGS	イメージをプッシュするときの **push** コマンドの追加パラメーター。	**string**	
SKIP_PUSH	コンテナーレジストリーへのイメージのプッシュをスキップします。	**string**	**false**
TLS_VERIFY	TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	詳細なロギングをオンにすると、実行されたすべてのコマンドがログに追加されます。	**string**	**false**

パラメーター	説明	型	デフォルト値
VERSION	言語バージョンに対応するイメージストリームのタグ。	**string**	**latest**

表3.45 s2i-ruby タスクでサポートされているワークスペース

ワークスペース	説明
source	S2I ワークフローのビルドコンテキストであるアプリケーションソースコード。
dockerconfig	Buildah がコンテナーレジストリーへのアクセスに使用する **.docker/config.json** ファイルを提供するためのオプションワークスペース。ファイルを **config.json** または **.dockerconfigjson** という名前のワークスペースの root に配置します。

表3.46 s2i-ruby タスクが返す結果

結果	型	説明
IMAGE_URL	**string**	ビルドされたイメージの完全修飾名。
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

skopeo-copy

skopeo-copy タスクは **skopeo copy** コマンドを実行します。

Skopeo は、リモートコンテナーイメージレジストリーを操作するためのコマンドラインツールであり、イメージのロードや実行にデーモンやその他のインフラストラクチャーを必要としません。**skopeo copy** コマンドは、別のリモートレジストリーにイメージをコピーします (たとえば、内部レジストリーから実稼働レジストリーにコピーします)。Skopeo は、ユーザーが提供する認証情報を使用したイメージレジストリーでの認可をサポートします。

skopeo-copy タスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-deploy-image
spec:
# ...
  tasks:
  - name: copy-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
```

```
    value: skopeo-copy
  - name: namespace
    value: openshift-pipelines
params:
- name: SOURCE_IMAGE_URL
  value: "docker://internal.registry/myimage:latest"
- name: DESTINATION_IMAGE_URL
  value: "docker://production.registry/myimage:v1.0"
workspaces:
- name: output
  workspace: shared-workspace
```

表3.47 skopeo-copy タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
SOURCE_IMAGE_URL	ソースコンテナーイメージのタグを含む完全修飾名。	**string**	
DESTINATION_IMAGE_URL	Skopeo がソースイメージをコピーする宛先イメージの、タグを含む完全修飾名。	**string**	
SRC_TLS_VERIFY	ソースレジストリーの TLS 検証フラグ。通常は **true** です。	**string**	**true**
DEST_TLS_VERIFY	宛先レジストリーの TLS 検証フラグ。通常は **true** です。	**string**	**true**
VERBOSE	デバッグ情報をログに出力します。	**string**	**false**

表3.48 skopeo-copy タスクでサポートされているワークスペース

ワークスペース	説明
images_url	複数のイメージをコピーする場合は、このワークスペースを使用してイメージの URL を指定します。

表3.49 skopeo-copy タスクが返す結果

結果	型	説明
SOURCE_DIGEST	**string**	ソースイメージの SHA256 ダイジェスト。
DESTINATION_DIGEST	**string**	宛先イメージの SHA256 ダイジェスト。

skopeo-copy ClusterTask からの変更点

- すべてのパラメーター名が大文字に変更されました。

- **VERBOSE** パラメーターが追加されました。
- ワークスペース名が **images-url** から **images_url** に変更されました。
- **SOURCE_DIGEST** および **DESTINATION_DIGEST** の結果が追加されました。

tkn
tkn タスクは、tkn を使用して Tekton リソースに対して操作を実行します。

tkn タスクの使用例

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
 name: tkn-run
spec:
 pipelineSpec:
   tasks:
   - name: tkn-run
     taskRef:
       resolver: cluster
       params:
       - name: kind
         value: task
       - name: name
         value: tkn
       - name: namespace
         value: openshift-pipelines
     params:
     - name: ARGS
```

表3.50 tkn タスクでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
SCRIPT	実行する **tkn** CLI スクリプト。	**string**	**tkn $@**
ARGS	実行する **tkn** CLI 引数。	**array**	**- --help**

表3.51 tkn タスクでサポートされているワークスペース

ワークスペース	説明
kubeconfig_dir	クラスターにアクセスするための認証情報を含む **.kube/config** ファイルを提供できるオプションのワークスペース。このファイルをワークスペースのルートに配置し、**kubeconfig** という名前を付けます。

tkn ClusterTask からの変更点

- **TKN_IMAGE** パラメーターが削除されました。
- ワークスペース名が **kubeconfig** から **kubeconfig_dir** に変更されました。

3.8. OPENSHIFT PIPELINES NAMESPACE で提供されるコミュニティータスク

デフォルトでは、OpenShift Pipelines のインストールには、パイプラインで使用できる一連のコミュニティータスクが含まれています。これらのタスクは、OpenShift Pipelines インストール namespace (通常は **openshift-pipelines** namespace) にあります。

argocd-task-sync-and-wait

argocd-task-sync-and-wait コミュニティータスクは、Argo CD アプリケーションをデプロイし、正常になるまで待機します。

これを行うには、以下の設定が必要です。* **argocd-env-configmap** config map で設定された Argo CD サーバーのアドレス。* **argocd-env-secret** シークレットに設定された認証情報。

アドレス情報を含む config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-env-configmap
data:
  ARGOCD_SERVER: https://argocd.example.com
# ...
```

認証情報を含むシークレットの例

```
apiVersion: v1
kind: Secret
metadata:
  name: argocd-env-secret
data:
  # Option 1
  ARGOCD_USERNAME: example_username (1)
  ARGOCD_PASSWORD: example_password
  # Option 2
  ARGOCD_AUTH_TOKEN: exmaple_token
```

(1) ユーザー名とパスワード、または認証トークンのいずれかを設定します。

argocd-task-sync-and-wait コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: argocd-task-sync-and-wait
spec:
  tasks:
    - name: argocd-task-sync-and-wait
      params:
        - name: application-name
          value: example_app_name
        - name: revision
          value: HEAD
```

```
        - name: flags
          value: '--'
        - name: argocd-version
          value: v2.2.2
      taskRef:
        kind: Task
        name: argocd-task-sync-and-wait
```

表3.52 argocd-task-sync-and-wait コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
application-name	デプロイするアプリケーションの名前。	
revision	デプロイするリビジョン。	**HEAD**
flags		**--**
argocd-version	Argo CD のバージョン。	**v2.2.2**

helm-upgrade-from-repo

helm -upgrade-from-repo コミュニティータスクは、指定された Helm リポジトリーとチャートに基づいて、OpenShift Container Platform クラスターに Helm チャートをインストールまたはアップグレードします。

Helm-upgrade-from-repo コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: helm-upgrade-from-repo
spec:
  tasks:
    - name: helm-upgrade-from-repo
      params:
        - name: helm_repo
          value: example_helm_repository
        - name: chart_name
          value: example_chart_name
        - name: release_version
          value: v1.0.0
        - name: release_name
          value: helm-release
        - name: release_namespace
          value: ''
        - name: overwrite_values
          value: ''
        - name: helm_image
          value: 'docker.io/lachlanevenson/k8s-
helm@sha256:5c792f29950b388de24e7448d378881f68b3df73a7b30769a6aa861061fd08ae'
      taskRef:
        kind: Task
        name: helm-upgrade-from-repo
```

表3.53 Helm-upgrade-from-repo コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
helm_repo	Helm リポジトリー。	
chart_name	デプロイする Helm チャート名。	
release_version	セマンティックバージョン管理形式の Helm リリースバージョン。	**v1.0.0**
release_name	Helm リリース名。	**helm-release**
release_namespace	Helm リリースの namespace。	**""**
overwrite_values	上書きする設定パラメーター (コンマ区切り)。例: **autoscaling.enabled=true,replicas=1**	**""**
helm_image	使用する Helm イメージ。	**docker.io/lachlanevenson/k8s-helm@sha256:5c792f29950b388de24e7448d378881f68b3df73a7b30769a6aa861061fd08ae**

helm-upgrade-from-source

Helm-upgrade-from-source コミュニティータスクは、指定されたチャートとソースワークスペースに基づいて、OpenShift Container Platform クラスターに Helm チャートをインストールおよびアップグレードします。

Helm-upgrade-from-source コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: helm-upgrade-from-source
spec:
  tasks:
    - name: helm-upgrade-from-source
      params:
        - name: charts_dir
          value: example_directory_path
        - name: release_version
          value: v1.0.0
        - name: release_name
          value: helm-release
        - name: release_namespace
          value: ''
        - name: overwrite_values
          value: ''
        - name: values_file
```

```
        value: values.yaml
      - name: helm_image
        value: 'docker.io/lachlanevenson/k8s-
helm@sha256:5c792f29950b388de24e7448d378881f68b3df73a7b30769a6aa861061fd08ae'
      - name: upgrade_extra_params
        value: ''
    taskRef:
      kind: Task
      name: helm-upgrade-from-source
    workspaces:
      - name: source
        workspace: shared-workspace
  #...
```

表3.54 Helm-upgrade-from-source コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
charts_dir	Helm チャートを含むソースワークスペース内のディレクトリー。	
release_version	セマンティックバージョン管理形式の Helm リリースバージョン。	**v1.0.0**
release_name	Helm リリース名。	**helm-release**
release_namespace	Helm リリースの namespace。	**""**
overwrite_values	上書きする設定パラメーター (コンマ区切り)。例: **autoscaling.enabled=true,replicas=1**	**""**
values_file	Helm の設定パラメーターを含むファイル。	**values.yaml**
helm_image	使用する Helm イメージ。	**docker.io/lachlanevenson/k8s-helm@sha256:5c792f29950b388de24e7448d378881f68b3df73a7b30769a6aa861061fd08ae**
upgrade_extra_params	Helm アップグレードコマンドに渡される追加のパラメーター。	**""**

表3.55 Helm-upgrade-from-source コミュニティータスクでサポートされているワークスペース

ワークスペース	説明
source	Helm チャートを含むワークスペース。

jib-maven

jib-maven コミュニティータスクは、Maven プロジェクト用の Jib ツールを使用して、Java、Kotlin、Groovy、および Scala ソースをコンテナーイメージにビルドします。

jib-maven コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: jib-maven
spec:
  tasks:
    - name: jib-maven
      params:
        - name: IMAGE
          value: example_image
        - name: MAVEN_IMAGE
          value: 'registry.redhat.io/ubi9/openjdk-
17@sha256:78613bdf887530100efb6ddf92d2a17f6176542740ed83e509cdc19ee7c072d6'
        - name: DIRECTORY
          value: .
        - name: CACHE
          value: empty-dir-volume
        - name: INSECUREREGISTRY
          value: 'false'
        - name: CACERTFILE
          value: service-ca.crt
      taskRef:
        kind: Task
        name: jib-maven
      workspaces:
        - name: source
          workspace: shared-workspace
  #...
```

表3.56 jib-maven コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
IMAGE	ビルドするイメージの名前。	
MAVEN_IMAGE	Maven ベースイメージ。	**registry.redhat.io/ubi9/openjdk-17@sha256:78613bdf887530100efb6ddf92d2a17f6176542740ed83e509cdc19ee7c072d6**
DIRECTORY	ソースリポジトリーのルートを基準とした、アプリケーションを含むディレクトリー。	**.**

パラメーター	説明	デフォルト値
CACHE	Maven アーティファクトとベースイメージレイヤーをキャッシュするためのボリュームの名前。	**empty-dir-volume**
INSECUREREGISTRY	安全でないレジストリーを許可します。	**false**
CACERTFILE	安全でないレジストリーサービスの認証局 (CA) バンドルファイル名。	**service-ca.crt**

表3.57 jib-maven コミュニティータスクでサポートされているワークスペース

ワークスペース	説明
source	Maven プロジェクトが含まれるワークスペース。
sslcertdir	SSL 証明書が含まれるオプションのワークスペース。

表3.58 jib-maven タスクが返す結果

結果	型	説明
IMAGE_DIGEST	**string**	ビルドされたイメージのダイジェスト。

jib-maven コミュニティークラスタータスクからの変更

- **IMAGE** および **MAVEN_IMAGE** パラメーターのデフォルト値が変更されました。

kubeconfig-creator

kubeconfig-creator コミュニティータスクは、パイプライン内の他のタスクがさまざまなクラスターにアクセスするために使用できる **kubeconfig** ファイルを作成します。

kubeconfig-creator コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: kubeconfig-creator
spec:
  tasks:
    - name: kubeconfig-creator
      params:
        - name: name
          value: example_cluster
        - name: url
          value: https://cluster.example.com
        - name: username
          value: example_username
        - name: password
```

```
        value: example_password
      - name: cadata
        value: ''
      - name: clientKeyData
        value: ''
      - name: clientCertificateData
        value: ''
      - name: namespace
        value: ''
      - name: token
        value: ''
      - name: insecure
        value: 'false'
    taskRef:
      kind: Task
      name: kubeconfig-creator
    workspaces:
      - name: output
        workspace: shared-workspace
#...
```

表3.59 kubeconfig-creator コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
name	アクセスするクラスターの名前。	
url	アクセスするクラスターのアドレス。	
username	クラスターへの Basic 認証用のユーザー名。	
password	クラスターへの Basic 認証用のパスワード。	""
cadata	PEM でエンコードされた認証局 (CA) 証明書。	""
clientKeyData	TLS のクライアントキーファイルからの PEM エンコードされたデータ。	""
clientCertificateData	TLS のクライアント証明書ファイルからの PEM エンコードされたデータ。	""
namespace	未指定のリクエストで使用するデフォルトの namespace。	""
token	クラスターへの認証用のベアラートークン。	""
insecure	TLS 証明書を検証せずにサーバーにアクセスするかどうかを示します。	**false**

表3.60 kubeconfig-creator コミュニティータスクでサポートされているワークスペース

ワークスペース	説明
output	**kubeconfig-creator** タスクが **kubeconfig** ファイルを保存するワークスペース。

pull-request

pull-request コミュニティータスクを使用すると、抽象化されたインターフェイスを通じてソースコントロール管理 (SCM) システムと対話できます。

このコミュニティータスクは、パブリック SCM インスタンスと、self-hosted またはエンタープライズ GitHub または GitLab インスタンスの両方で機能します。

ダウンロードモードでは、このタスクは、**.MANIFEST** ファイルを含む既存のプルリクエストの状態を **pr** ワークスペースに入力します。

アップロードモードでは、このタスクは **.MANIFEST** ファイルを含む **pr** ワークスペースの内容をプルリクエストの内容と比較し、内容が異なる場合は、**pr** ワークスペースと一致するようにプルリクエストを更新します。

pull-request コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pull-request
spec:
spec:
  tasks:
    - name: pull-request
      params:
        - name: mode
          value: upload
        - name: url
          value: https://github.com/example/pull/xxxxx
        - name: provider
          value: github
        - name: secret-key-ref
          value: example_secret
        - name: insecure-skip-tls-verify
          value: 'false'
      taskRef:
        kind: Task
        name: pull-request
      workspaces:
        - name: pr
          workspace: shared-workspace
  #...
```

表3.61 pull-request コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値

パラメーター	説明	デフォルト値
mode	**download** に設定すると、**url** のプルリクエストの状態が取得されます。**upload** に設定すると、**URL** のプルリクエストが更新されます。	
url	プルリクエストの URL。	
provider	SCM システムのタイプ。サポートされる値は **github** または **gitlab** です。	
secret-key-ref	base64 でエンコードされた SCM トークンを含む **token** と呼ばれるキーを含む、**Opaque** タイプの **Secret** オブジェクトの名前。	
insecure-skip-tls-verify	**true** に設定すると、証明書の検証は無効になります。	**false**

表3.62 pull-request コミュニティータスクでサポートされているワークスペース

ワークスペース	説明
pr	プルリクエストの状態が含まれるワークスペース。

trigger-jenkins-job

curl リクエストを使用して Jenkins ジョブをトリガーするには、 **trigger-jenkins-job** コミュニティータスクを使用できます。

trigger-jenkins-job コミュニティータスクの使用例

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: trigger-jenkins-job
spec:
  tasks:
    - name: trigger-jenkins-job
      params:
        - name: JENKINS_HOST_URL
          value: example_host_URL
        - name: JOB_NAME
          value: example_job_name
        - name: JENKINS_SECRETS
          value: jenkins-credentials
        - name: JOB_PARAMS
          value:
            - example_param
      taskRef:
        kind: Task
```

```
      name: trigger-jenkins-job
    workspaces:
      - name: source
        workspace: shared-workspace
# ...
```

表3.63 trigger-jenkins-job コミュニティータスクでサポートされているパラメーター

パラメーター	説明	デフォルト値
JENKINS_HOST_URL	Jenkins が実行されているサーバー URL。	
JOB_NAME	トリガーする必要がある Jenkins ジョブ。	
JENKINS_SECRETS	認証情報を含む Jenkins シークレット。	**jenkins-credentials**
JOB_PARAMS	**curl** リクエストの一部として追加する引数。	""

表3.64 trigger-jenkins-job コミュニティータスクでサポートされているワークスペース

ワークスペース	説明
source	**curl** リクエストを通じて Jenkins ジョブに送信できるファイルをマウントするために使用できるワークスペース。

3.9. OPENSHIFT PIPELINES で提供されるステップアクション定義

OpenShift Pipelines は、タスクで使用できる標準の **StepAction** 定義を提供します。これらの定義を参照するには、クラスターリゾルバーを使用します。

git-clone

git-clone ステップアクションは Git を使用してワークスペースでリモートリポジトリーを初期化し、クローンします。このステップアクションを使用して、このソースコードをビルドまたは処理するパイプラインの開始時にリポジトリーのクローンを作成するタスクを定義できます。

タスクでの git-clone ステップアクションの使用例

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: clone-repo-anon
spec:
# ...
  steps:
  - name: clone-repo-step
    ref:
      resolver: cluster
      params:
      - name: name
        value: git-clone
```

```
    - name: namespace
      value: openshift-pipelines
    - name: kind
      value: stepaction
  params:
  - name: URL
    value: $(params.url)
  - name: OUTPUT_PATH
    value: $(workspaces.output.path)
```

表3.65 git-clone ステップアクションでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
OUTPUT_PATH	フェッチされた Git リポジトリーのディレクトリー。クローンされたリポジトリーデータは、ディレクトリーのルートまたは **SUBDIRECTORY** パラメーターで定義された相対パスに配置されます。	**string**	
SSH_DIRECTORY_PATH	必要に応じて、秘密鍵、**known_hosts**、**config** などのファイルを含む **.ssh** ディレクトリー。このディレクトリーを指定すると、タスクは Git リポジトリーへの認証にそれを使用します。認証情報を安全に保存するために、このディレクトリーを提供するワークスペースを **Secret** リソースにバインドします。	**string**	
BASIC_AUTH_PATH	**.gitconfig** および **.git-credentials** ファイルを含むディレクトリー。このディレクトリーを指定すると、タスクはそのリポジトリーを Git リポジトリーへの認証に使用します。可能な場合は、認証に **BASIC_AUTH_PATH** ではなく **SSH_DIRECTORY_PATH** ディレクトリーを使用します。認証情報を安全に保存するために、このディレクトリーを提供するワークスペースを **Secret** リソースにバインドします。	**string**	
SSL_CA_DIRECTORY_PATH	CA 証明書を含むワークスペース。このワークスペースを指定すると、Git は HTTPS を使用してリモートリポジトリーとやり取りする場合に、これらの証明書を使用してピアを検証します。	**string**	
CRT_FILENAME	**ssl-ca-directory** ワークスペース内の認証局 (CA) のバンドルファイル名。	**string**	**ca-bundle.crt**
HTTP_PROXY	HTTP プロキシーサーバー (TLS 以外のリクエスト)。	**string**	

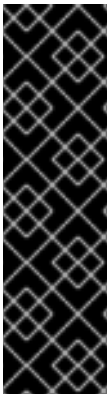
パラメーター	説明	型	デフォルト値
HTTPS_PROXY	HTTPS プロキシーサーバー (TLS リクエスト)。	**string**	
NO_PROXY	HTTP/HTTPS リクエストのプロキシーをオプトアウトします。	**string**	
SUBDIRECTORY	タスクが Git リポジトリーを配置する **output** ワークスペース内の相対パス。	**string**	
USER_HOME	Pod 内の Git ユーザーのホームディレクトリーへの絶対パス。	**string**	**/home/git**
DELETE_EXISTING	Git 操作を実行する前に、デフォルトのワークスペースにコンテンツが存在する場合は削除します。	**string**	**true**
VERBOSE	実行したコマンドをログに記録します。	**string**	**false**
SSL_VERIFY	グローバル **http.sslVerify** 値。リモートリポジトリーを信頼していない限り、このパラメーターを **false** に設定しないでください。	**string**	**true**
URL	Git リポジトリー URL	**string**	
リビジョン	チェックアウトするリビジョン (ブランチやタグなど)。	**string**	**main**
REFSPEC	リビジョンをチェックアウトする前にタスクが取得するリポジトリーの **refspec** 文字列。	**string**	
SUBMODULES	Git サブモジュールを初期化して取得します。	**string**	**true**
DEPTH	取得するコミットの数。"シャロークローン" は単一のコミットです。	**string**	**1**
SPARSE_CHECKOUT_DIRECTORIES	"スパースチェックアウト" を実行するための、コンマで区切られたディレクトリーパターンのリスト。	**string**	

表3.66 git-clone ステップのアクションが返す結果

結果	型	説明

結果	型	説明
COMMIT	**string**	クローンされた Git リポジトリー内の現在のブランチの HEAD にあるコミットの SHA ダイジェスト。
URL	**string**	クローンされたリポジトリーの URL。
COMMITTER_DATE	**string**	クローンされた Git リポジトリー内の現在のブランチの HEAD にあるコミットのエポックタイムスタンプ。

cache-upload および cache-fetch



重要

cache-upload および **cache-fetch** ステップアクションの使用は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品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ことがあります。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。

ビルドプロセスが依存関係を保持するキャッシュディレクトリーを保持するには、**cache-upload** および **cache-fetch** ステップアクションを使用し、Amazon Simple Storage Service (S3) バケット、Google Cloud Services (GCS) バケット、または Open Container Initiative (OCI) リポジトリーに保存します。

cache-upload ステップアクションを使用すると、ステップアクションはビルド内の特定のファイルに基づいてハッシュを計算します。これらのファイルを選択するには、正規表現を指定する必要があります。**cache-upload** ステップアクションは、ハッシュでインデックス付けされたキャッシュディレクトリーのコンテンツを含むイメージを保存します。

cache-fetch ステップアクションを使用すると、ステップアクションは同じハッシュを計算します。次に、このハッシュのキャッシュされたイメージがすでに利用可能かどうかを確認します。イメージが利用可能な場合、ステップアクションによってキャッシュされたコンテンツがキャッシュディレクトリーに設定されます。イメージが利用できない場合は、ディレクトリーはそのまま残ります。

cache-fetch ステップアクションを使用した後、ビルドプロセスを実行できます。キャッシュが正常に取得された場合、ビルドプロセスで以前にダウンロードした依存関係が含まれます。キャッシュが取得されなかった場合、ビルドプロセスの通常の手順で依存関係をダウンロードします。

cache-fetch の結果は、キャッシュされたイメージが取得されたかどうかを示します。後続の **cache-upload** ステップアクションでは、その結果を使用して、現在のハッシュのキャッシュがすでに利用可能であった場合は、新しいキャッシュイメージのアップロードをスキップできます。

次のサンプルタスクは、リポジトリーからソースを取得し、キャッシュ (使用可能な場合) を取得し、Maven ビルドを実行し、キャッシュが取得されなかった場合は、ビルドディレクトリーの新しいキャッシュイメージをアップロードします。

タスク内の cache-fetch および cache-upload ステップアクションの使用例

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: java-demo-task
spec:
  workspaces:
  - name: source
  params:
  - name: repo_url
    type: string
    default: https://github.com/sample-organization/sample-java-project.git
  - name: revision
    type: string
    default: main
  - name: registry
    type: string
    default: image-registry.openshift-image-registry.svc:5000/sample-project/mvn-cache
  - name: image
    type: string
    default: openjdk:latest
  - name: buildCommand
    type: string
    default: "maven -Dmaven.repo.local=${LOCAL_CACHE_REPO} install"
  - name: cachePatterns
    type: array
    default: ["**pom.xml"]
  - name: force-cache-upload
    type: string
    default: "false"
  steps:
   - name: create-repo
     image: $(params.image)
     script: |
       mkdir -p $(workspaces.source.path)/repo
       chmod 777 $(workspaces.source.path)/repo
   - name: fetch-repo
     ref:
       resolver: cluster
       params:
       - name: name
         value: git-clone
       - name: namespace
         value: openshift-pipelines
       - name: kind
         value: stepaction
     params:
       - name: OUTPUT_PATH
         value: $(workspaces.source.path)/repo
       - name: URL
         value: $(params.repo_url)
       - name: REVISION
         value: $(params.revision)
   - name: cache-fetch
     ref:
```

```
      resolver: cluster
      params:
      - name: name
        value: cache-fetch
      - name: namespace
        value: openshift-pipelines
      - name: kind
        value: stepaction
    params:
    - name: PATTERNS
      value: $(params.cachePatterns)
    - name: SOURCE
      value: oci://$(params.registry):{{hash}}
    - name: CACHE_PATH
      value: $(workspaces.source.path)/cache
    - name: WORKING_DIR
      value: $(workspaces.source.path)/repo
  - name: run-build
    image: $(params.image)
    workingDir: $(workspaces.source.path)/repo
    env:
      - name: LOCAL_CACHE_REPO
        value: $(workspaces.source.path)/cache/repo
    script: |
      set -x
      $(params.buildCommand)
      echo "Cache size is $(du -sh $(workspaces.source.path)/cache)"
  - name: cache-upload
    ref:
      resolver: cluster
      params:
      - name: name
        value: cache-upload
      - name: namespace
        value: openshift-pipelines
      - name: kind
        value: stepaction
    params:
      - name: PATTERNS
        value: $(params.cachePatterns)
      - name: TARGET
        value: oci://$(params.registry):{{hash}}
      - name: CACHE_PATH
        value: $(workspaces.source.path)/cache
      - name: WORKING_DIR
        value: $(workspaces.source.path)/repo
      - name: FORCE_CACHE_UPLOAD
        value: $(params.force-cache-upload)
```

表3.67 cache-fetch ステップアクションでサポートされているパラメーター

パラメーター	説明	型	デフォルト値

パラメーター	説明	型	デフォルト値
パターン	ハッシュを計算するファイルを選択するための正規表現。たとえば、Go プロジェクトの場合、キャッシュを計算するために **go.mod** ファイルを使用できます。この場合、このパラメーターの値は ****/go.sum** になります (** はサブディレクトリーを表します (階層は問わない))。	**array**	
ソース	キャッシュを取得するためのソース URI。キャッシュハッシュを指定するには **{{hash}}** を使用します。サポートされているタイプは、**oci** (例: **oci://quay.io/example-user/go-cache:{{hash}}**) と **s3** (例: **s3://example-bucket/{{hash}}**) です。	**string**	
CACHE_PATH	キャッシュコンテンツを抽出するためのパス。通常、このパスはワークスペース内にあります。	**string**	
WORKING_DIR	ハッシュを計算するためのファイルが配置されているパス。	**string**	
INSECURE	**"true"** の場合、キャッシュの取得に安全でないモードを使用します。	**string**	**"false"**
GOOGLE_APPLICATION_CREDENTIALS	Google 認証情報が保存されているパス。空の場合は無視されます。	**string**	
AWS_CONFIG_FILE	AWS 設定ファイルへのパス。空の場合は無視されます。	**string**	
AWS_SHARED_CREDENTIALS_FILE	AWS 認証情報ファイルへのパス。空の場合は無視されます。	**string**	
BLOB_QUERY_PARAMS	S3、GCS、または Azure を設定するための Blob クエリーパラメーター。S3 アクセラレーション、FIPS、パススタイルのアドレス指定などの追加機能には、これらのオプションパラメーターを使用します。	**string**	

表3.68 cache-fetch ステップアクションが返す結果

結果	型	説明
fetched	**string**	ステップがキャッシュをフェッチした場合は **"true"**、ステップがキャッシュをフェッチしていない場合は **"false"**。

表3.69 **cache-upload** ステップアクションでサポートされているパラメーター

パラメーター	説明	型	デフォルト値
パターン	ハッシュを計算するファイルを選択するための正規表現。たとえば、Go プロジェクトの場合、キャッシュを計算するために **go.mod** ファイルを使用できます。この場合、このパラメーターの値は ****/go.sum** になります (** はサブディレクトリーを表します (階層は問わない))。	**array**	
TARGET	キャッシュをアップロードするためのターゲット URI。**{{hash}}** を使用してキャッシュハッシュを指定します。サポートされているタイプは、**oci** (例: **oci://quay.io/example-user/go-cache:{{hash}}**) と **s3** (例: **s3://example-bucket/{{hash}}**) です。	**string**	
CACHE_PATH	ステップがイメージにパックするキャッシュコンテンツのパス。通常、このパスはワークスペース内にあります。	**string**	
WORKING_DIR	ハッシュを計算するためのファイルが配置されているパス。	**string**	
INSECURE	**"true"** の場合、キャッシュのアップロードに安全でないモードを使用します。	**string**	**"false"**
FETCHED	**"true"** の場合、このハッシュのキャッシュはすでに取得されています。	**string**	**"false"**
FORCE_CACHE_UPLOAD	**"true"** の場合、このステップでは、以前に取得されたキャッシュであってもアップロードされます。	**string**	**"false"**
GOOGLE_APPLICATION_CREDENTIALS	Google 認証情報が保存されているパス。空の場合は無視されます。	**string**	
AWS_CONFIG_FILE	AWS 設定ファイルへのパス。空の場合は無視されます。	**string**	

パラメーター	説明	型	デフォルト値
AWS_SHARED_CREDENTIALS_FILE	AWS 認証情報ファイルへのパス。空の場合は無視されます。	**string**	
BLOB_QUERY_PARAMS	S3、GCS、または Azure を設定するための Blob クエリーパラメーター。S3 アクセラレーション、FIPS、パススタイルのアドレス指定などの追加機能には、これらのオプションパラメーターを使用します。	**string**	

cache-upload ステップアクションは結果を返しません。

3.10. バージョン付けされていないタスクとバージョン付けされたタスクとステップアクションについて

openshift-pipelines namespace には、バージョン付けされたタスクとステップアクションおよび標準のバージョン付けされていないのタスクとステップアクションが含まれます。たとえば、Red Hat OpenShift Pipelines Operator バージョン 1.18 をインストールすると、次のアイテムが作成されます。

- **buildah-1-18-0** バージョン付けされたタスク
- **buildah** バージョン付けされていないタスク
- **git-clone-1-18-0** バージョン付けされた **StepAction** 定義
- **git-clone** バージョン付けされていない **StepAction** 定義

バージョン付けされていないタスクおよびバージョン付けされたタスクおよびステップアクションには、**params**、**workspaces**、および **steps** など、同じメタデータ、動作、仕様があります。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。

バージョン付けされていないタスクまたはバージョン付けされたタスクおよびステップアクション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3.70 バージョン付けされていないタスクとバージョン付けされたタスクとステップアクションの長所と短所

	メリット	デメリット

	メリット	デメリット
バージョン付けされていないタスクおよびステップアクション	● 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、バージョン付けされていないタスクとステップアクションを使用します。 ● 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タスクおよびステップアクション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タスクおよびステップアクションよりも少ないリソースを消費します。	● バージョン付けされていないタスクおよびステップアクション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タスクとステップアクションが後方互換性を持たない場合、Operator のアップグレード後にパイプラインが破損する可能性があります。
バージョン付けされたタスクおよびステップアクション	● バージョン更新後も変更されない実稼働環境のパイプラインを優先する場合は、バージョン管理されたタスクとステップアクションを使用します。 ● Operator の新しいバージョンをインストールすると、現在のマイナーバージョンとその直前のマイナーバージョンのバージョン付きタスクとステップアクションが保持されます。	● 以前のバージョン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。 ● アップグレード後、Operator は、以前のバージョン付けされたタスクとステップアクションを管理できません。以前のバージョンを手動で削除した場合、復元できません。 ● アップグレード後、Operator は以前のマイナーリリースより前のバージョンからバージョン管理されたタスクとステップアクションを削除できます。新しいバージョンをインストールし、以前のバージョンのバージョン付きタスクまたはステップアクションを削除すると、以前のバージョンのバージョン付きタスクを使用するパイプラインは動作しなくなります。

バージョン付けされていないタスクとバージョン付けされたタスクとステップアクションにはさまざまな命名規則があり、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3.71 バージョン付けされていないタスクとバージョン付けされたタスクとステップアクションの違い

	命名法	アップグレード
バージョン付けされていないタスクおよびステップアクション	バージョン付けされていないタスクおよびステップアクションには、タスクまたはステップアクションの名前のみが含まれます。たとえば、Operator v1.18 でインストールされる Buildah のバージョン付けされていないタスクの名前は **buildah** です。	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タスクとステップアクションが最新の変更で更新されます。名前は変更されません。
バージョン付けされたタスクおよびステップアクション	バージョン付けされたタスクおよびステップアクションには、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 v1.18 でインストールされた Buildah のバージョン付きタスクの名前は **buildah-1-18-0** です。	Operator をアップグレードすると、バージョン管理されたタスクとステップアクションの最新バージョンがインストールされ、直前のバージョンが保持され、以前のバージョンが削除されます。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 1.18 をインストールすると、**buildah-1-18-0** タスクがインストールされ、**buildah-1-17-0** タスクは保持され、**buildah-1-16-0** などの以前のバージョンは削除されます。

3.11. 関連情報

- OpenShift Pipelines での Tekton Hub の使用

第4章 OPENSHIFT PIPELINES での手動承認の使用

パイプラインで手動承認タスクを指定できます。パイプラインがこのタスクに到達すると、一時停止し、OpenShift Container Platform ユーザー 1 つ以上からの承認を待機します。いずれかのユーザーがタスクを承認せずに拒否した場合、パイプラインは失敗します。この機能は、manual approval gate controller によって提供されます。

重要

manual approval gate はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品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ことがあります。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。

4.1. MANUAL APPROVAL GATE CONTROLLER の有効化

手動承認タスクを使用するには、まず manual approval gate controller 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。
- **oc** コマンドラインユーティリティーを使用してクラスターにログオンしている。
- **openshift-pipelines** namespace の管理者権限がある。

手順

1. **ManualApprovalGate** カスタムリソース (CR) の次のマニフェストを含めて、 **manual-approval-gate-cr.yaml** という名前のファイルを作成します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: ManualApprovalGate
metadata:
  name: manual-approval-gate
spec:
  targetNamespace: openshift-pipelines
```

2. 以下のコマンドを実行して、**ManualApprovalGate** CR を適用します。

```
$ oc apply -f manual-approval-gate-cr.yaml
```

3. 次のコマンドを入力して、manual approval gate controller が実行されていることを確認します。

```
$ oc get manualapprovalgates.operator.tekton.dev
```

出力例

```
NAME                  VERSION    READY   REASON
manual-approval-gate   v0.1.0     True
```

READY ステータスが **True** であることを確認します。**True** でない場合は、数分待ってからもう一度コマンドを実行します。コントローラーが準備完了状態になるまでには、しばらく時間がかかる場合があります。

4.2. 手動承認タスクの指定

パイプラインで手動承認タスクを指定できます。パイプライン実行のタスク実行がこのタスクに到達すると、パイプライン実行は停止し、1 つ以上のユーザーからの承認を待機します。

前提条件

- manual approver gate controller を有効にしている。
- パイプラインの YAML 仕様を作成している。

手順

- 以下の例のように、パイプラインに **ApprovalTask** を指定します。

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: example-manual-approval-pipeline
spec:
  tasks:
# ...
  - name: example-manual-approval-task
    taskRef:
      apiVersion: openshift-pipelines.org/v1alpha1
      kind: ApprovalTask
    params:
    - name: approvers
      value:
      - user1
      - user2
      - user3
    - name: description
      value: Example manual approval task - please approve or reject
    - name: numberOfApprovalsRequired
      value: '2'
    - name: timeout
      value: '60m'
# ...
```

表4.1 手動承認タスクのパラメーター

パラメーター	型	説明
approvers	array	タスクを承認できる OpenShift Container Platform ユーザー。

パラメーター	型	説明
description	string	オプション: 承認タスクの説明。OpenShift Pipelines は、タスクを承認または拒否できるユーザーに説明を表示します。
numberOfApprovalsRequired	string	タスクに必要なさまざまなユーザーからの承認の数。
timeout	string	オプション: 承認のタイムアウト期間。この期間中にタスクが指定の数の承認が受信されなかった場合、パイプラインの実行は失敗します。デフォルトのタイムアウトは 1 時間です。

4.3. 手動承認タスクの承認

承認タスクを含むパイプラインを実行し、実行が承認タスクに到達すると、パイプラインの実行は一時停止し、ユーザーの承認または拒否を待機します。

ユーザーは、Web コンソールまたは **opc** コマンドラインユーティリティーを使用してタスクを承認または拒否できます。

タスクに設定されている承認者のいずれかがタスクを拒否した場合、パイプラインの実行は失敗します。

1 つのユーザーがタスクを承認したが、設定された承認数にまだ達していない場合は、同じユーザーがタスクを拒否するように変更できるようになっており、パイプラインの実行は失敗します。

4.3.1. Web コンソールを使用した手動承認タスクの承認

OpenShift Container Platform Web コンソールを使用して、手動承認タスクを承認または拒否できます。

手動承認タスクの承認者としてリストされており、パイプライン実行がこのタスクに到達すると、Web コンソールに通知が表示されます。承認が必要なタスクのリストを表示し、これらのタスクを承認または拒否できます。

前提条件

- OpenShift Pipelines コンソールプラグインを有効にしている。

手順

1. 次のいずれかのアクションを完了して、承認できるタスクのリストを表示します。
   - 承認が必要なタスクに関する通知が表示されたら、この通知の **Go to Approvals タブ** をクリックします。

- Administrator パースペクティブメニューで、Pipelines → Pipelines を選択してから Approvals タブをクリックします。
- Developer パースペクティブメニューで、Pipelines を選択し、Approvals タブをクリックします。
- PipelineRun details ウィンドウの Details タブで、手動承認タスクを表す四角形をクリックします。リストには、このタスクの承認のみが表示されます。
- PipelineRun details ウィンドウで、ApprovalTasks タブをクリックします。リストには、このパイプライン実行の承認のみが表示されます。

2. 承認タスクのリストで、承認するタスクを表す行で、 アイコンをクリックし、次のいずれかのオプションを選択します。
   - タスクを承認するには、Approve を選択します。
   - タスクを拒否するには、Reject を選択します。
3. Reason フィールドにメッセージを入力します。
4. Submit をクリックします。

関連情報

- OpenShift Pipelines コンソールプラグインの有効化

4.3.2. コマンドラインを使用した手動承認タスクの承認

opc コマンドラインユーティリティーを使用して、手動承認タスクを承認または拒否できます。自分が承認者であるタスクのリストを表示し、承認待ちのタスクを承認または拒否できます。

前提条件

- **opc** コマンドラインユーティリティーをダウンロードしてインストールしている。このユーティリティーは、**tkn** コマンドラインユーティリティーと同じパッケージで使用できます。
- **oc** コマンドラインユーティリティーを使用してクラスターにログオンしている。

手順

1. 次のコマンドを入力して、自分が承認者としてリストされている手動承認タスクのリストを表示します。

```
$ opc approvaltask list
```

出力例

```
NAME                                    NumberOfApprovalsRequired   PendingApprovals   Rejected
STATUS
manual-approval-pipeline-01w6e1-task-2  2                           0                  0        Approved
```

```
manual-approval-pipeline-6ywv82-task-2   2                 2                 0                 Rejected
manual-approval-pipeline-90gyki-task-2   2                 2                 0                 Pending
manual-approval-pipeline-jyrkb3-task-2   2                 1                 1                 Rejected
```

2. オプション: 手動承認タスクに関する情報 (名前、namespace、パイプライン実行名、承認者のリスト、現在のステータスなど) を表示するには、次のコマンドを入力します。

```
$ opc approvaltask describe <approval_task_name>
```

3. 必要に応じて手動承認タスクを承認または拒否します。

- 手動承認タスクを承認するには、以下のコマンドを入力します。

```
$ opc approvaltask approve <approval_task_name>
```

  オプションで、**-m** パラメーターを使用して承認のメッセージを指定できます。

```
$ opc approvaltask approve <approval_task_name> -m <message>
```

- 手動承認タスクを拒否するには、次のコマンドを入力します。

```
$ opc approvaltask reject <approval_task_name>
```

  オプションで、**-m** パラメーターを使用して拒否のメッセージを指定できます。

```
$ opc approvaltask reject <approval_task_name> -m <message>
```

関連情報

- tkn のインストール

第5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は、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。

5.1. 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 で実行し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
```

5.2. 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、**openshift-pipelines** namespace で提供されている **buildah** タスクまたはこのタスクのコピーを使用して、次の例に示すように **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:
          resolver: cluster
          params:
          - name: kind
            value: task
          - name: name
            value: buildah
          - name: namespace
            value: openshift-pipelines
        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

5.3. 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。

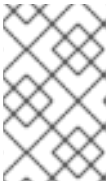
```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-rhel-entitlement
  namespace: <pipeline_namespace> 1
rules:
  - apiGroups:
      - sharedresource.openshift.io
    resources:
      - sharedsecrets
    resourceNames:
      - shared-rhel-entitlement
    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、**openshift-pipelines** namespace で提供されている **buildah** タスクまたはこのタスクのコピーを使用して、次の例に示すように **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:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name
          value: buildah
        - name: namespace
          value: openshift-pipelines
      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. パイプライン実行における **rhel-entitlement** ワークスペースの定義。**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

5.4. 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