OpenShift Container Platform 4.5

Pipeline

OpenShift Container Platform での Pipeline の設定および使用

Last Updated: 2021-07-13

OpenShift Container Platform 4.5 Pipeline

OpenShift Container Platform での Pipeline の設定および使用

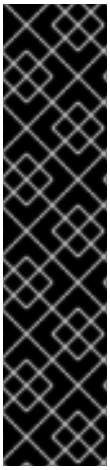
法律上の通知

概要

本書では、OpenShift Container Platform でパイプラインを設定し、使用する方法を説明します。

目次

第1章 OPENSHIFT PIPELINE について



重要

OpenShift Pipelines は、テクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

Red Hat OpenShift Pipelines は、Kubernetes リソースをベースとしたクラウドネイティブの継続的インテグレーションおよび継続的デリバリー (CI/CD) ソリューションです。これは Tekton ビルディングブロックを使用し、基礎となる実装の詳細を抽象化することで、複数のプラットフォームでのデプロイメントを自動化します。Tekton では、Kubernetes ディストリビューション間で移植可能な CI/CD パイプラインを定義するための標準のカスタムリソース定義 (CRD、Customer Resource Definition) が多数導入されています。

1.1. 主な特長

- Red Hat OpenShift Pipelines は、分離されたコンテナーで必要なすべての依存関係と共に Pipeline を実行するサーバーレスの CI/CD システムです。
- Red Hat OpenShift Pipelines は、マイクロサービスベースのアーキテクチャーで機能する分散型チーム向けに設計されています。
- Red Hat OpenShift Pipelines は、拡張および既存の Kubernetes ツールとの統合を容易にする標準の CI/CD パイプライン定義を使用し、オンデマンドのスケーリングを可能にします。
- Red Hat OpenShift Pipelines を使用して、Kubernetes プラットフォーム全体で移植可能な S2I (Source-to-Image)、Buildah、Buildpacks、および Kaniko などの Kubernetes ツールを使用してイメージをビルドできます。
- OpenShift Container Platform Developer Console を使用して、Tekton リソースの作成、Pipeline 実行のログの表示、OpenShift Container Platform namespace でのパイプラインの管理を実行できます。

1.2. RED HAT OPENSHIFT PIPELINES の概念

Red Hat OpenShift Pipelines は、アプリケーション用に CI/CD パイプラインをアセンブルできるビルディングブロックとして動作する標準のカスタムリソース定義 (CRD) のセットを提供します。

Task

Task は Pipeline の設定可能な最小単位です。これは基本的に Pipeline のビルドを構成する入出力の機能です。これは個別に実行することも、Pipeline の一部として実行することもできます。Pipeline には 1 つまたは複数の Task が含まれており、各 Task は 1 つまたは複数のステップで構成されます。ステップは、Task によって順次実行される一連のコマンドです。

Pipeline

Pipeline は、アプリケーションのビルド、デプロイメント、およびデリバリーを自動化する複雑なワークフローを構築するために実行される一連の Task で構成されます。これは、PipelineResource、パラメーターおよび 1 つまたは複数の Task のコレクションです。Pipeline は、Task に入力および出力として追加される PipelineResource を使用して外部と対話します。

PipelineRun

PipelineRun は、Pipeline の実行中のインスタンスです。PipelineRun は Pipeline を開始し、Pipeline で実行される各 Task の TaskRun の作成を管理します。

TaskRun

TaskRun は、Pipeline の各 Task について PipelineRun によって自動的に作成されます。これは Pipeline で Task のインスタンスを実行する結果です。また、Task が Pipeline の外部で実行される場合に手動で作成できます。

Workspace

Workspace は、Task がランタイム時に入力を受信するか、または出力を提供するために必要となるストレージボリュームです。Task または Pipeline は Workspace を宣言し、TaskRun または PipelineRun は、宣言された Workspace にマウントされるストレージボリュームの実際の場所を提供します。これにより、Task が柔軟かつ再利用可能となり、複数の Task 間で Workspace を共有できます。

Trigger

Trigger は Git プル要求などの外部イベントを取得し、イベントペイロードを処理して情報の主要な部分を抽出します。次にこの抽出された情報は一連の事前定義済みのパラメーターにマップされます。これにより、Kubernetes リソースの作成およびデプロイメントが含まれる可能性のある一連のタスクがトリガーされます。Trigger を Pipeline と共に使用して、実行が完全に Kubernetes リソースで定義される本格的な CI/CD システムを作成できます。

Condition

Condition は、Task が Pipeline で実行される前に、実行される検証またはチェックのことを指します。Condition は、論理テストを実行する **if** ステートメントと似ています。戻り値は **True** または **False** です。Task は、すべての Condition が **True** を返す場合に実行されますが、いずれかの Condition が失敗すると、Task と後続のすべての Task は省略されます。Pipeline で Condition を使用して、複数のシナリオに対応する複雑なワークフローを作成できます。

1.3. 詳細な OPENSHIFT PIPELINES の概念

本書では、Pipeline の各種概念を詳述します。

1.3.1. Task

Task は Pipeline のビルディングブロックであり、順次実行される Step で構成されます。これらは再利用可能であり、複数の Pipeline で使用することができます。

Steps は、イメージのビルドなど、特定の目的を達成するための一連のコマンドです。各 Task は Pod として実行され、各 Step は同じ Pod 内の独自のコンテナーで実行されます。Step は同じ Pod 内で実行されるため、ファイル、ConfigMap、およびシークレットをキャッシュするために同じボリュームにアクセスできます。

以下の例は、**apply-manifests** Task を示しています。

```
apiVersion: tekton.dev/v1beta1 (1)
kind: Task (2)
metadata:
  name: apply-manifests (3)
spec: (4)
```

```
  params:
  - default: k8s
    description: The directory in source that contains yaml manifests
    name: manifest_dir
    type: string
  steps:
  - args:
    - |-
      echo Applying manifests in $(inputs.params.manifest_dir) directory
      oc apply -f $(inputs.params.manifest_dir)
      echo -----------------------------------
    command:
    - /bin/bash
    - -c
    image: quay.io/openshift/origin-cli:latest
    name: apply
    workingDir: /workspace/source
  workspaces:
  - name: source
```

1. Task API バージョン **v1beta1**。
2. Kubernetes オブジェクトのタイプを指定します。この例では、**Task** です。
3. この Task の一意の名前。
4. Task のパラメーターおよびステップと、Task によって使用される Workspace を一覧表示します。

この Task は Pod を開始し、**maven:3.6.0-jdk-8-slim** イメージ内でコンテナーを実行して指定されたコマンドを実行します。アプリケーションのソースコードが含まれる **workspace-git** という入力ディレクトリーを受信します。

Task は Git リポジトリーのプレースホルダーのみを宣言し、使用する Git リポジトリーを指定しません。これにより、Task を複数の Pipeline および目的のために再利用可能にできます。

1.3.2. TaskRun

TaskRun は、クラスター上の特定の入出力、および実行パラメーターで実行するために Task をインスタンス化します。これは独自に起動することも、PipelineRun の一部として起動することもできます。

Task はコンテナーイメージを実行する 1 つ以上の Step で構成され、各コンテナーイメージは特定のビルド作業を実行します。TaskRun は、すべての Step が正常に実行されるか、または失敗が発生するまで、指定された順序で Task の Step を実行します。

以下の例は、関連する入力パラメーターで **apply-manifests** Task を実行する TaskRun を示しています。

```
apiVersion: tekton.dev/v1beta1 1
kind: TaskRun 2
metadata:
  name: apply-manifests-taskrun 3
spec: 4
  serviceAccountName: pipeline
```

```
  taskRef: 5
    kind: Task
    name: apply-manifests
  workspaces: 6
  - name: source
    persistentVolumeClaim:
      claimName: source-pvc
```

1. TaskRun API バージョン **v1beta1**
2. Kubernetes オブジェクトのタイプを指定します。この例では、**TaskRun** です。
3. この TaskRun を識別する一意の名前。
4. TaskRun の定義。この TaskRun には、Task と必要な Workspace を指定します。
5. この TaskRun に使用される Task 参照の名前。この TaskRun は Task **apply-manifests** Task を実行します。
6. TaskRun によって使用される Workspace。

1.3.3. Pipeline

Pipeline は、特定の実行順序で編成される Task のコレクションです。1 つ以上の Task を含む Pipeline を使用して、アプリケーションの CI/CD ワークフローを定義できます。

Pipeline 定義は、多くのフィールドまたは属性で構成され、Pipeline が特定の目的を達成することを可能にします。各 Pipeline 定義には、特定の入力を取り込み、特定の出力を生成する Task が少なくとも 1 つ含まれる必要があります。Pipeline 定義には、アプリケーション要件に応じて Condition、Workspace、Parameter、または Resource をオプションで含めることもできます。

以下の例は、**buildah** ClusterTask を使用して Git リポジトリーからアプリケーションイメージをビルドする **build-and-deploy** Pipeline を示しています。

```
apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "release-tech-preview-2"
  - name: IMAGE
    type: string
```

```
  description: image to be built from the code
tasks: 7
- name: fetch-repository
  taskRef:
    name: git-clone
    kind: ClusterTask
  workspaces:
  - name: output
    workspace: shared-workspace
  params:
  - name: url
    value: $(params.git-url)
  - name: subdirectory
    value: ""
  - name: deleteExisting
    value: "true"
  - name: revision
    value: $(params.git-revision)
- name: build-image 8
  taskRef:
    name: buildah
    kind: ClusterTask
  params:
  - name: TLSVERIFY
    value: "false"
  - name: IMAGE
    value: $(params.IMAGE)
  workspaces:
  - name: source
    workspace: shared-workspace
  runAfter:
  - fetch-repository
- name: apply-manifests 9
  taskRef:
    name: apply-manifests
  workspaces:
  - name: source
    workspace: shared-workspace
  runAfter: 10
  - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  workspaces:
  - name: source
    workspace: shared-workspace
  params:
  - name: deployment
    value: $(params.deployment-name)
  - name: IMAGE
    value: $(params.IMAGE)
  runAfter:
  - apply-manifests
```

1 Pipeline API バージョン **v1beta1**。

2 Kubernetes オブジェクトのタイプを指定します。この例では、**Pipeline** です。

3 この Pipeline の一意の名前。

4 Pipeline の定義および構造を指定します。

5 Pipeline のすべての Task で使用される Workspace。

6 Pipeline のすべての Task で使用されるパラメーター。

7 Pipeline で使用される Task の一覧を指定します。

8 Task **build-image**: **buildah** ClusterTask を使用して、所定の Git リポジトリーからアプリケーションイメージをビルドします。

9 Task **apply-manifests**: 同じ名前のユーザー定義 Task を使用します。

10 Task が Pipeline で実行されるシーケンスを指定します。この例では、**apply-manifests** Task は **build-image** Task の完了後にのみ実行されます。

1.3.4. PipelineRun

PipelineRun は、クラスター上の特定の入力、出力、および実行パラメーターで実行される Pipeline をインスタンス化します。対応する TaskRun は、PipelineRun の Task ごとに自動的に作成されます。

Pipeline のすべての Task は、すべての Task が正常に実行されるか、または Task が失敗するまで定義されたシーケンスで実行されます。**status** フィールドは、監視および監査のために、PipelineRun で各 TaskRun の進捗を追跡し、保存します。

以下の例は、関連するリソースおよびパラメーターで **build-and-deploy** Pipeline を実行する PipelineRun を示しています。

```
apiVersion: tekton.dev/v1beta1 1
kind: PipelineRun 2
metadata:
  name: build-deploy-api-pipelinerun 3
spec:
  pipelineRef:
    name: build-and-deploy 4
  params: 5
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: http://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    persistentvolumeclaim:
      claimName: source-pvc
```

1 PipelineRun API バージョン **v1beta1**。

2 Kubernetes オブジェクトのタイプを指定します。この例では、**PipelineRun** です。

3. この PipelineRun を識別する一意の名前。
4. 実行する Pipeline の名前。この例では、**build-and-deploy** です。
5. Pipeline の実行に必要なパラメーターの一覧を指定します。
6. PipelineRun によって使用される Workspace。

1.3.5. Workspace

注記

PipelineResource はデバッグが容易ではなく、スコープの制限があり、Task を再利用可能にしないため、OpenShift Pipelines では PipelineResource の代わりに Workspace を使用することが推奨されます。

Workspace は、Pipeline の Task がランタイム時に必要とする共有ストレージボリュームを宣言します。Workspace では、ボリュームの実際の場所を指定する代わりに、ランタイム時に必要となるファイルシステムまたはファイルシステムの一部を宣言できます。TaskRun または PipelineRun で Workspace にマウントされるボリュームの特定の場所の詳細を指定する必要があります。ランタイムストレージボリュームからボリューム宣言を分離することで、Task を再利用可能かつ柔軟にし、ユーザー環境から切り離すことができます。

Workspace を使用すると、以下が可能になります。

- Task の入力および出力の保存
- Task 間でのデータの共有
- Secret に保持される認証情報のマウントポイントとして使用
- ConfigMap に保持される設定のマウントポイントとして使用
- 組織が共有する共通ツールのマウントポイントとして使用
- ジョブを高速化するビルドアーティファクトのキャッシュの作成

以下を使用して、TaskRun または PipelineRun で Workspace を指定できます。

- 読み取り専用 ConfigMap または Secret
- 他の Task と共有される既存の PersistentVolumeClaim
- 指定された VolumeClaimTemplate からの PersistentVolumeClaim
- TaskRun の完了時に破棄される emptyDir

以下の例は、Pipeline で定義される、**build-image** および **apply-manifests** Task の **shared-workspace** Workspace を宣言する **build-and-deploy** Pipeline のコードスニペットを示しています。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
```

```
  workspaces: (1)
  - name: shared-workspace
  params:
...
  tasks: (2)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces: (3)
    - name: source (4)
      workspace: shared-workspace (5)
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: (6)
    - name: source
      workspace: shared-workspace
    runAfter:
     - build-image
...
```

(1) Pipeline で定義される Task 間で共有される Workspace の一覧。Pipeline は、必要な数の Workspace を定義できます。この例では、**shared-workspace** という名前の 1 つの Workspace のみが宣言されます。

(2) Pipeline で使用される Task の定義。このスニペットは、共通の Workspace を共有する **build-image** および **apply-manifests** の 2 つの Task を定義します。

(3) **build-image** Task で使用される Workspace の一覧。Task 定義には、必要な数の Workspace を含めることができます。ただし、Task が最大 1 つの書き込み可能な Workspace を使用することが推奨されます。

(4) Task で使用される Workspace を一意に識別する名前。この Task は、**source** という名前の 1 つの Workspace を使用します。

(5) Task によって使用される Pipeline Workspace の名前。Workspace **source** は Pipeline Workspace の **shared-workspace** を使用することに注意してください。

(6) **apply-manifests** Task で使用される Workspace の一覧。この Task は、 **build-image** Task と **source** Workspace を共有することに注意してください。

以下は、**build-deploy-api-pipelinerun** PipelineRun のコードスニペットです。これは、**build-and-deploy** Pipeline で使用される **shared-workspace** Workspace のストレージボリュームを定義するために PersistentVolumeClaim を使用します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
```

```
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
  params:
...

  workspaces: (1)
  - name: shared-workspace (2)
    persistentvolumeclaim:
      claimName: source-pvc (3)
```

(1) PipelineRun にボリュームバインディングを提供する Pipeline Workspace の一覧を指定します。

(2) ボリュームが提供されている Pipeline の Workspace の名前。

(3) Workspace に割り当てられる事前に定義された PersistentVolumeClaim の名前を指定します。この例では、既存の **source-pvc** PersistentVolumeClaim が **shared-workspace** Workspace に割り当てられています。

1.3.6. Trigger

Trigger を Pipeline と併用して、Kubernetes リソースで CI/CD 実行全体を定義する本格的な CI/CD システムを作成します。Pipeline の Trigger は外部イベントをキャプチャーし、それらのイベントを処理して情報の主要な部分を抽出します。このイベントデータを事前に定義されたパラメーターのセットにマップすると、Kubernetes リソースを作成し、デプロイできる一連のタスクがトリガーされます。

たとえば、アプリケーションの Red Hat OpenShift Pipelines を使用して CI/CD ワークフローを定義します。アプリケーションリポジトリーで新たな変更を有効にするには、PipelineRun を開始する必要があります。Trigger は変更イベントをキャプチャーし、処理することにより、また新規イメージを最新の変更でデプロイする PipelineRun をトリガーして、このプロセスを自動化します。

Trigger は、再利用可能で分離した自律型 CI/CD システムを構成するように連携する以下の主なコンポーネントで構成されています。

- **EventListener** は、JSON ペイロードを含む受信 HTTP ベースイベントをリッスンするエンドポイントまたはイベントシンクを提供します。EventListener は、Event インターセプターを使用してペイロードで軽量イベント処理を実行します。これはペイロードのタイプを特定し、オプションでこれを変更します。現在、Pipeline Trigger は Webhook インターセプター、GitHub インターセプター、GitLab インターセプター、および Common Expression Language (CEL) インターセプターの 4 種類のインターセプターをサポートします。

- **TriggerBinding** は、イベントペイロードからフィールドを抽出し、それらをパラメーターとして保存します。

- **TriggerTemplate** は、TriggerBinding からパラメーター化されたデータを使用する方法を指定します。TriggerTemplate は、TriggerBinding から入力を受信し、新規 PipelineResources の作成および新規 PipelineRun の開始につながる一連のアクションを実行するリソーステンプレートを定義します。

EventListener は、TriggerBinding と TriggerTemplate の概念を関連付けます。EventListener は受信イベントをリッスンし、インターセプターを使用して基本的なフィルターを処理し、TriggerBinding を使用してデータを抽出してから、TriggerTemplate を使用して Kubernetes リソースを作成するためにデータを処理します。

以下の例は、**vote-app-binding** TriggerBinding のコードスニペットを示しています。これは、受信イベントペイロードから Git リポジトリー情報を抽出します。

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: TriggerBinding (2)
metadata:
  name: vote-app (3)
spec:
  params: (4)
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

(1) TriggerBinding API バージョン **v1alpha1**。

(2) Kubernetes オブジェクトのタイプを指定します。この例では、**TriggerBinding** です。

(3) この TriggerBinding を識別する一意の名前。

(4) 受信イベントペイロードから抽出され、TriggerTemplate に渡されるパラメーターの一覧。この例では、Git リポジトリー URL、名前、およびリビジョンはイベントペイロードの本体から抽出されます。

以下の例は **vote-app-template** TriggerTemplate のコードスニペットを示しています。これは、TriggerBinding から受信される Git リポジトリー情報から Pipeline リソースを作成します。

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: TriggerTemplate (2)
metadata:
  name: vote-app (3)
spec:
  params: (4)
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: (5)
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(tt.params.git-repo-name)-$(uid)
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
```

```
      value: $(tt.params.git-repo-name)
    - name: git-url
      value: $(tt.params.git-repo-url)
    - name: git-revision
      value: $(tt.params.git-revision)
    - name: IMAGE
      value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(tt.params.git-repo-name)
  workspaces:
  - name: shared-workspace
    persistentvolumeclaim:
      claimName: source-pvc
```

1. TriggerTemplate API バージョン **v1alpha1**。
2. Kubernetes オブジェクトのタイプを指定します。この例では、**TriggerTemplate** です。
3. この TriggerTemplate を識別する一意の名前。
4. TriggerBinding または EventListerner によって提供されるパラメーター。
5. TriggerBinding または EventListener で受信されるパラメーターから Pipeline 用に作成されるリソーステンプレートの一覧。

以下の例は、受信イベントを処理するために **vote-app-binding** TriggerBinding および **vote-app-template** TriggerTemplate を使用する EventListener を示しています。

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: EventListener (2)
metadata:
  name: vote-app (3)
spec:
  serviceAccountName: pipeline (4)
  triggers:
  - bindings: (5)
    - ref: vote-app
    template: (6)
      name: vote-app
```

1. EventListener API バージョン **v1alpha1**。
2. Kubernetes オブジェクトのタイプを指定します。この例では、**EventListener** です。
3. この EventListener を識別する一意の名前。
4. 使用されるサービスアカウント名。
5. この EventListener に使用される TriggerBinding の名前。
6. この Eventlistener に使用される Triggertemplate の名前。

1.4. 追加リソース

- Pipeline のインストールについての詳細は、「OpenShift Pipeline のインストール」を参照してください。
- カスタムの CI/CD ソリューションの作成についての詳細は、「CI/CD パイプライン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。

第2章 OPENSHIFT PIPELINE のインストール

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできること。
- **oc** CLI がインストールされていること。
- OpenShift Pipeline (**tkn**) CLI がローカルシステムにインストールされていること。

2.1. WEB コンソールでの RED HAT OPENSHIFT PIPELINES OPERATOR のインストール

OpenShift Container Platform OperatorHub に一覧表示されている Operator を使用して Red Hat OpenShift Pipelines をインストールできます。Red Hat OpenShift Pipelines Operator をインストールする際に、Pipeline の設定に必要なカスタムリソース (CR) は Operator と共に自動的にインストールされます。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → OperatorHub** に移動します。
2. **Filter by keyword** ボックスを使用して、カタログで **Red Hat OpenShift Pipelines Operator** を検索します。**OpenShift Pipelines Operator** タイルをクリックします。

注記

OpenShift Pipelines Operator の **コミュニティー** バージョンを選択しないようにしてください。

3. **Red Hat OpenShift Pipelines Operator** ページで Operator についての簡単な説明を参照してください。**Install** をクリックします。
4. **Install Operator** ページで以下を行います。
   a. **Installation Mode** について **All namespaces on the cluster (default)**を選択します。このモードは、デフォルトの **openshift-operators** namespace で Operator をインストールし、Operator がクラスターのすべての namespace を監視し、これらの namespace に対して利用可能になるようにします。
   b. **Approval Strategy** について **Automatic** を選択します。これにより、Operator への今後のアップグレードは Operator Lifecycle Manager (OLM) によって自動的に処理されます。**Manual** 承認ストラテジーを選択すると、OLM は更新要求を作成します。クラスター管理者は、Operator を新規バージョンに更新できるように OLM 更新要求を手動で承認する必要があります。
   c. **Update Channel** を選択します。
      - **ocp-<4.x>** チャネルは、 Red Hat OpenShift Pipelines Operator の最新の安定したリリースのインストールを可能にします。

- **preview** チャネルは、Red Hat OpenShift Pipelines Operator の最新プレビューバージョンのインストールを有効にします。これには、4.x 更新チャネルでは利用できない機能が含まれる場合があります。

5. **Install** をクリックします。Operator が **Installed Operators** ページに一覧表示されます。

注記

Operator は **openshift-operators** namespace に自動的にインストールされます。

6. **Status** が **Succeeded Up to date** に設定され、Red Hat OpenShift Pipelines Operator のインストールが正常に行われたことを確認します。

2.2. CLI を使用した OPENSHIFT PIPELINES OPERATOR のインストール

CLI を使用して OperatorHub から Red Hat OpenShift Pipelines Operator をインストールできます。

手順

1. Subscription オブジェクトの YAML ファイルを作成し、namespace を Red Hat OpenShift Pipelines Operator にサブスクライブします (例: **sub.yaml**)。

Subscription の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel:  <channel name> 1
  name: openshift-pipelines-operator-rh 2
  source: redhat-operators 3
  sourceNamespace: openshift-marketplace 4
```

1. Operator のサブスクライブ元のチャネル名を指定します。
2. サブスクライブする Operator の名前。
3. Operator を提供する CatalogSource の名前。
4. CatalogSource の namespace。デフォルトの OperatorHub CatalogSource には **openshift-marketplace** を使用します。

2. Subscription オブジェクトを作成します。

```
$ oc apply -f sub.yaml
```

Red Hat OpenShift Pipelines Operator がデフォルトのターゲット namespace **openshift-operators** にインストールされるようになりました。

追加リソース

- Operator の OpenShift Container Platform へのインストール方法については、「Operator のクラスターへの追加」セクションを参照してください。

第3章 OPENSHIFT PIPELINE のアンインストール

Red Hat OpenShift Pipelines Operator のアンインストールは 2 つの手順で実行されます。

1. Red Hat OpenShift Pipelines Operator のインストール時にデフォルトで追加されたカスタムリソース (CR) を削除します。
2. Red Hat OpenShift Pipelines Operator をアンインストールします。

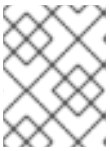
Operator のみをアンインストールしても、Operator のインストール時にデフォルトで作成される Red Hat OpenShift Pipelines コンポーネントは削除されません。

3.1. RED HAT OPENSHIFT PIPELINES コンポーネントおよびカスタムリソースの削除

Red Hat OpenShift Pipelines Operator のインストール時にデフォルトで作成されるカスタムリソース (CR) を削除します。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Administration → Custom Resource Definition** に移動します。
2. **Filter by name** ボックスに **config.operator.tekton.dev** を入力し、 Red Hat OpenShift Pipelines Operator CR を検索します。
3. **CRD Config** をクリックし、 **Custom Resource Definition Details**ページを表示します。
4. **Actions** ドロップダウンメニューをクリックし、**Delete Custom Resource Definition**を選択します。



注記

CR を削除すると Red Hat OpenShift Pipelines コンポーネントが削除され、クラスター上のすべての Task および Pipeline が失われます。

5. **Delete** をクリックし、CR の削除を確認します。

3.2. RED HAT OPENSHIFT PIPELINES OPERATOR のアンインストール

手順

1. **Operators → OperatorHub** ページから、**Filter by keyword** ボックスを使用して **Red Hat OpenShift Pipelines Operator** を検索します。
2. **OpenShift Pipelines Operator** タイルをクリックします。Operator タイルはこれがインストールされていることを示します。
3. **OpenShift Pipelines Operator** 記述子ページで、**Uninstall** をクリックします。

追加リソース

- Operator の OpenShift Container Platform でのアンインストール方法については、「クラスターからの Operator の削除」セクションを参照してください。

第4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 Task を作成するか、既存の再利用可能な Task をインストールします。
- アプリケーションの配信 Pipeline を作成し、定義します。
- Workspace に割り当てられる PersistentVolumeClaim を作成し、Pipeline 実行のボリュームまたはファイルシステムを提供します。
- PipelineRun を作成して、Pipeline をインスタンス化し、これを起動します。
- Trigger を追加し、ソースリポジトリーのイベントを取得します。

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

- フロントエンドインターフェース **vote-ui** および **ui-repo** Git リポジトリーにあるソースコード。
- バックエンドインターフェース **vote-api**、および **api-repo** Git リポジトリーにあるソースコード。
- **pipelines-tutorial** Git リポジトリーにある **apply_manifest** および **update-deployment** Task。

4.1. 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 Pipeline をインストールしている。インストールが完了すると、クラスター全体に適用可能になります。
- 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 **ui-repo** およびバックエンド **api-repo** Git リポジトリーをフォークしており、管理者がこれらのリポジトリーにアクセスできる。
- 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。

4.2. プロジェクトの作成および PIPELINE SERVICEACCOUNT 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** ServiceAccount 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 ServiceAccount を追加し、設定します。この ServiceAccount は PipelineRun によって使用されます。

```
$ oc get serviceaccount pipeline
```

4.3. PIPELINE TASK の作成

手順

1. **pipelines-tutorial** リポジトリーから **apply-manifests** および **update-deployment** Task をインストールします。これには、Pipeline の再利用可能な Task の一覧が含まれます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/02_update_deployment_task.yaml
```

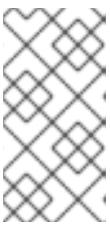
2. **tkn task list** コマンドを使用して、作成した Task を一覧表示します。

```
$ tkn task list
```

出力では、**apply-manifests** および **update-deployment** Task が作成されていることを検証します。

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

3. **tkn clustertasks list** コマンドを使用して、--**buildah** および **s2i-python-3** などの Operator でインストールされた追加の ClusterTask を一覧表示します。



注記

特権付きセキュリティーコンテキストが必要になるため、特権付き Pod コンテナーを使用して **buildah** ClusterTask を実行する必要があります。Pod の SCC (Security Context Constraints) についての詳細は、「追加リソース」セクションを参照してください。

```
$ tkn clustertasks list
```

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

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

4.4. PIPELINE のアセンブル

Pipeline は CI/CD フローを表し、実行する Task 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。

Pipeline は、**from** および **runAfter** パラメーターを使用して Task 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、Pipeline の各 Task が実行中に必要な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 Pipeline を作成します。

Pipeline 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の **vote-ui** について以下のタスクを実行します。

- **git-url** および **git-revision** パラメーターを参照して、Git リポジトリーからアプリケーションのソースコードのクローンを作成します。
- **buildah** ClusterTask 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内部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** Task 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 Pipeline YAML ファイルの内容をコピーし、保存します。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
  - name: shared-workspace
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "release-tech-preview-2"
```

```
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

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

2. Pipeline 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/release-tech-preview-2/01_pipeline/04_pipeline.yaml
```

3. **tkn pipeline list** コマンドを使用して、Pipeline 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** Pipeline 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   ---        ---       ---        ---
```

4.5. WORKSPACE での PERSISTENTVOLUMECLAIM の VOLUMESOURCE としての指定

Workspace は Task がデータを共有する際に使用でき、これにより Pipeline の各 Task が実行時に必要となる 1 つまたは複数のボリュームを指定することができます。

このセクションでは、データストレージを提供し、これを Workspace にバインドするために PersistentVolumeClaim を作成します。この PersistentVolumeClaim は、Pipeline の実行に必要なボリュームまたはファイルシステムを提供します。

手順

1. 以下の PersistentVolumeClaim YAML ファイルのサンプルをコピーし、保存します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

2. 作成したファイルを指定して PersistentVolumeClaim を作成します。

```
$ oc create -f <PersistentVolumeClaim-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ができます。

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/03_persistent_volume_claim.yaml
```

4.6. PIPELINE の実行

PipelineRun は Pipeline 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、Pipeline の各タスクについて TaskRun を自動的に作成し、開始します。

手順

1. バックエンドアプリケーションの Pipeline を起動します。

```
$ tkn pipeline start build-and-deploy -w name=shared-workspace,claimName=source-pvc -p deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-api.git -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
```

   コマンド出力で返される PipelineRun ID をメモします。

2. PipelineRun の進捗を追跡します。

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

3. フロントエンドアプリケーションの Pipeline を起動します。

```
$ tkn pipeline start build-and-deploy -w name=shared-workspace,claimName=source-pvc -p deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-ui.git -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-ui
```

   コマンド出力で返される PipelineRun ID をメモします。

4. PipelineRun の進捗を追跡します。

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

5. 数分後に、**tkn pipelinerun list** コマンドを使用して、すべての PipelineRun を一覧表示して Pipeline が正常に実行されたことを確認します。

```
$ tkn pipelinerun list
```

   出力には、PipelineRun が一覧表示されます。

```
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 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 Pipeline の PipelineResource および ServiceAccount を使用して最後の PipelineRun を再実行するには、以下を実行します。

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

4.7. TRIGGER の PIPELINE への追加

Trigger は、Pipeline 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 Pipeline をアセンブルし、起動した後に、TriggerBinding、TriggerTemplate、および EventListener 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/v1alpha1
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** を作成します。

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

または、**TriggerBinding** 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/release-tech-preview-2/03_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/v1alpha1
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: release-tech-preview-2
  - name: git-repo-name
    description: The name of the deployment to be created / patched
```

```
  resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(params.git-repo-name)-$(uid)
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/$(tt.params.git-repo-name)
      workspaces:
      - name: shared-workspace
        persistentvolumeclaim:
          claimName: source-pvc
```

4. **TriggerTemplate** を作成します。

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

または、**TriggerTemplate** 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/release-tech-
preview-2/03_triggers/02_template.yaml
```

5. 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/v1alpha1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
    template:
      name: vote-app
```

6. **EventListener** を作成します。

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

または、**EvenListener** 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/release-tech-preview-2/03_triggers/03_event_listener.yaml
```

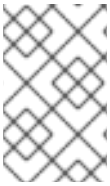
7. EventListener サービスを OpenShift Container Platform ルートとして公開し、これをアクセス可能にします。

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

4.8. 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 EventListener によって受信される HTTP POST メッセージです。その後、イベントペイロードは TriggerBinding にマップされ、TriggerTemplate によって処理されます。TriggerTemplate は最終的に 1 つ以上の PipelineRun 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 **vote-ui** および **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。

```
$ 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 **vote-ui** をブラウザーで開きます。
   b. **Settings → Webhooks → Add Webhook** をクリックします。
   c. **Webhooks/Add Webhook** ページで以下を実行します。
      i. 手順 1 の Webhook URL を **Payload URL** フィールドに入力します。
      ii. **Content type** について **application/json** を選択します。
      iii. シークレットを **Secret** フィールドに指定します。
      iv. **Just the push event**が選択されていることを確認します。
      v. **Active** を選択します。
      vi. **Add Webhook** をクリックします。
3. バックエンドリポジトリー **vote-api** について手順 2 を繰り返します。

4.9. PIPELINERUN のトリガー

push イベントが Git リポジトリーで実行されるたびに、設定された Webhook はイベントペイロードを公開される EventListener サービスルートに送信します。アプリケーションの EventListener サービスはペイロードを処理し、これを関連する TriggerBinding と TriggerTemplate のペアに渡します。TriggerBinding はパラメーターを抽出し、TriggerTemplate はこれらのパラメーターを使用してリソースを作成します。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 **vote-ui** リポジトリーにプッシュし、PipelineRun 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 **vote-ui** のクローンを作成します。

```
$ git clone git@github.com:<your GitHub ID>/vote-ui.git -b release-tech-preview-2
```

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 release-tech-preview-2
```

3. PipelineRun がトリガーされたかどうかを確認します。

```
$ tkn pipelinerun list
```

新規の PipelineRun が開始されたことに注意してください。

4.10. 追加リソース

- **Developer** パースペクティブについての詳細は、「**Developer** パースペクティブでの Pipeline の使用」セクションを参照してください。
- SCC (Security Context Constraints) の詳細は、「Managing Security Context Constraints」セクションを参照してください。
- 再利用可能な Task の追加の例については、OpenShift Catalog リポジトリーを参照してください。さらに、Tekton プロジェクトで Tekton Catalog を参照することもできます。

第5章 DEVELOPER パースペクティブを使用した RED HAT OPENSHIFT PIPELINES の使用

OpenShift Container Platform Web コンソールの **Developer** パースペクティブを使用して、ソフトウェア配信プロセスの CI/CD パイプラインを作成できます。

Developer パースペクティブ:

- **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 Pipeline を作成します。
- **Add → From Git** オプションを使用して、OpenShift Container Platform でアプリケーションを作成する間に Operator によってインストールされた Pipeline テンプレートおよびリソースを使用して Pipeline を作成します。

アプリケーションの Pipeline の作成後に、**Pipelines** ビューでデプロイされた Pipeline 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 Pipeline と対話することもできます。**Pipeline Builder** を使用して作成された Pipeline を **Topology** ビューに表示するには、カスタムラベルをこの Pipeline に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、Web コンソールで Developer パースペクティブに切り替えていること。
- クラスターに OpenShift Pipelines Operator がインストールされていること。
- クラスター管理者か、または create および edit パーミッションを持つユーザーであること。
- プロジェクトを作成していること。

5.1. 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。

- 既存の Task および ClusterTask を使用して Pipeline 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 Pipeline ClusterTask をクラスターに追加します。
- Pipeline Run 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 Pipeline に追加します。
- Pipeline の各 Task のこれらの Pipeline リソースを入力および出力リソースとして参照します。
- Task 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。必要な場合は、Task の Pipeline に追加される追加のパラメーターを参照します。

手順

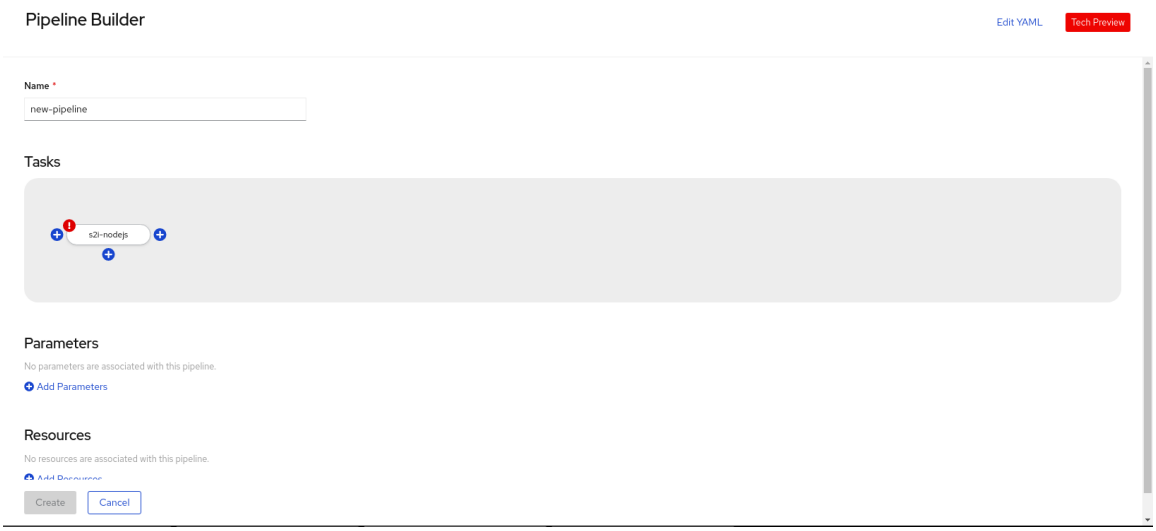
1. **Developer** パースペクティブの **Add** ビューで、**Pipeline** タイルをクリックし、**Pipeline Builder** ページを表示します。
2. Pipeline の一意の名前を入力します。

3. **Select task** 一覧から Task を選択し、Task を Pipeline に追加します。この例では、 **s2i-nodejs** Task を使用します。

   a. 連続する Task を Pipeline に追加するには、Task の右側または左側にあるプラスアイコンをクリックし、**Select task** 一覧から、Pipeline に追加する必要のある Task を選択します。この例では、**s2i-nodejs** Task の右側にあるプラスアイコンを使用して、 **openshift-client** Task を追加します。

   b. 並列の Task を既存の Task に追加するには、Task の下に表示されるプラスアイコンをクリックし、**Select Task** 一覧から Pipeline に追加する必要のある並列 Task を選択します。

   図5.1 Pipeline Builder



4. **Add Resources** をクリックし、Pipeline Run が使用する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、Pipeline の Task 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。

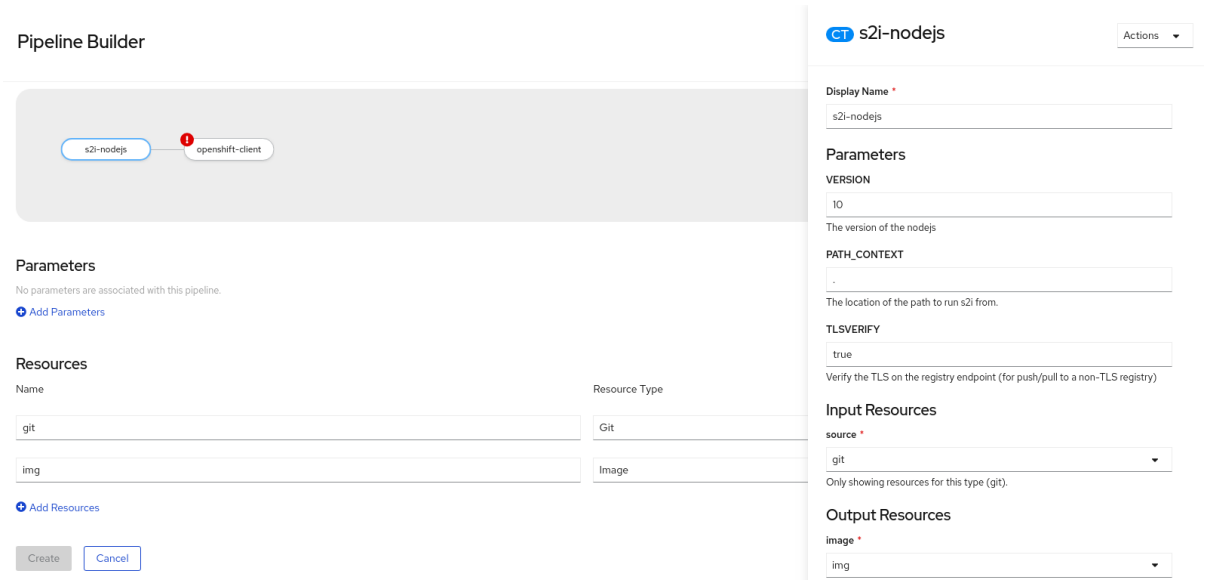
   a. 入力リソースを追加します。**Name** フィールドに **Source** を入力し、**Resource Type** ドロップダウンリストから **Git** を選択します。

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

5. Task 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。必要な場合は、 **Add Parameters** リンクを使用してパラメーターを追加します。

6. Task のリソースが指定されていない場合、**Missing Resources** (リソース不足) の警告が Task に表示されます。**s2i-nodejs** Task をクリックし、Task の詳細情報が含まれるサイドパネルを表示します。

図5.2 Pipelines Builder の Task の詳細



7. Task サイドパネルで、そのリソースおよびパラメーターを指定します。

    a. **Input Resources → Source** セクションで、**Select Resources** ドロップダウンリストに、Pipeline に追加したリソースが表示されます。この例では、**Source** を選択します。

    b. **Output Resources → Image** セクションで **Select Resources** リストをクリックし、**Img** を選択します。

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

8. 同様に、**openshift-client** Task の入力リソースを追加します。

9. **Create** をクリックして Pipeline を作成します。作成された Pipeline の詳細を表示する **Pipeline Details** ページにリダイレクトされます。**Action** ボタンを使用して Pipeline を開始できるようになりました。

オプションで、**Pipeline Builder** ページの右上にある **Edit YAML** リンクを使用して、コンソールで Pipeline YAML ファイルを直接変更することもできます。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 Pipeline を作成することもできます。

5.2. OPENSHIFT PIPELINE を使用したアプリケーションの作成

アプリケーションと共に Pipeline 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。詳細は、「Creating applications using the Developer perspective」を参照してください。

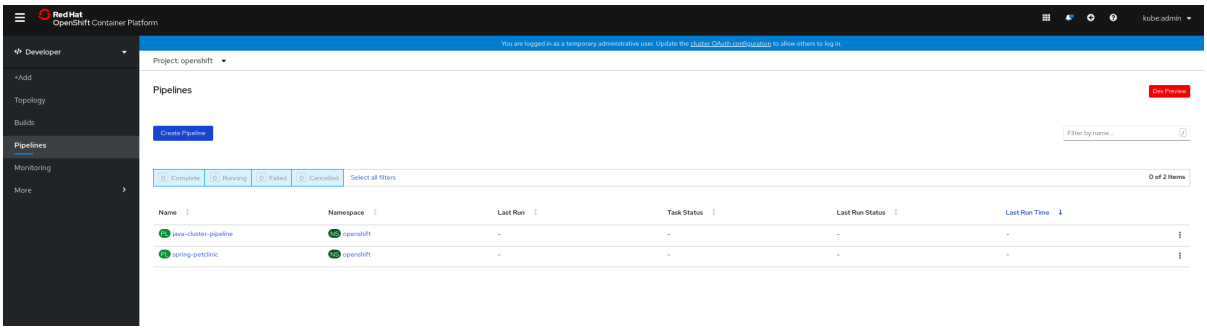
5.3. DEVELOPER パースペクティブを使用した PIPELINE の使用

Developer パースペクティブの **Pipelines** ビューは、プロジェクトのすべての Pipeline を、Pipeline が作成された namespace、最後の PipelineRun、PipelineRun の Task のステータス、および実行にかかった時間などの詳細と共に一覧表示します。

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 Pipeline を表示します。

図5.3 Developer パースペクティブの Pipeline ビュー



2. 必要な Pipeline をクリックし、**Pipeline Details** ページを表示します。このページには、Pipeline のすべての直列および並列 Task が視覚的に表示されます。Task はページの右下にも一覧表示されます。一覧表示されている **Task** をクリックし、Task の詳細を表示できます。

3. オプションで、**Pipeline Details** ページで以下を実行します。

   - **YAML** タブをクリックし、Pipeline の YAML ファイルを編集します。
   - **Pipeline Runs** タブをクリックして、Pipeline の完了済み、実行中、または失敗した実行を確認します。Options メニュー を使用して、実行中の Pipeline を停止するか、以前の Pipeline 実行と同じパラメーターとリソースを使用して Pipeline を再実行するか、または PipelineRun を削除します。
   - **Parameters** タブをクリックして、Pipeline 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、または編集することもできます。
   - **Resources** タブをクリックして、Pipeline で定義されたリソースを表示します。必要に応じて追加のリソースを追加するか、または編集することもできます。

5.4. PIPELINE の起動

Pipeline の作成後に、これを開始し、これに含まれる Task 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。Pipeline Run 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 Pipeline 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、Pipeline 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、Pipeline 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 Pipeline の場合、**Start Pipeline** ダイアログボックスでは **Parameters** セクションに **APP_NAME** フィールドも表示され、ダイアログボックスのすべてのフィールドが Pipeline テンプレートによって事前に入力されます。

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

3. オプション: **Advanced Options** を変更し、認証情報を追加して、指定された Git サーバーまたは Docker 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 Crendentials** については、認証する **Registry Server Address** を指定し、**Username**、**Password**、および **Email** フィールドに認証情報を指定します。
        追加の **Registry Server Address** を指定する必要がある場合は、**Add Credentials** を選択します。

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

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

      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。

   Pipeline 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 PipelineRun を開始します。

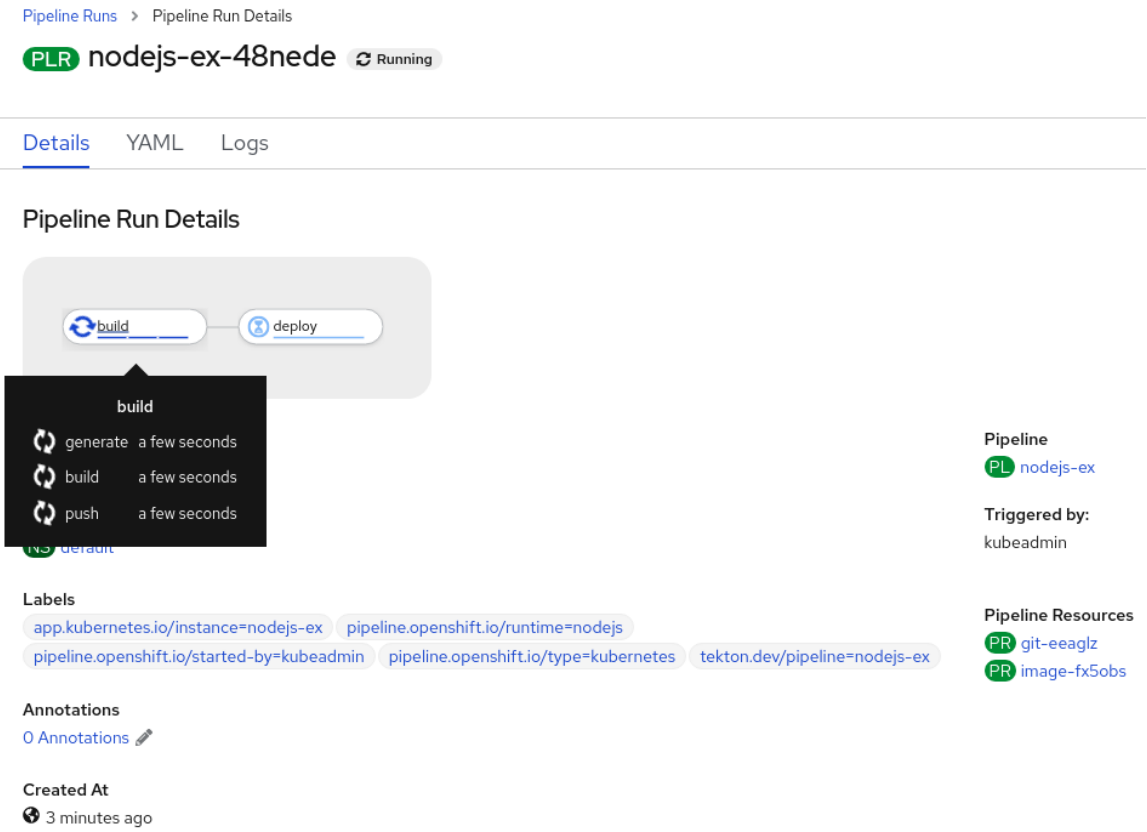
5. **Pipeline Run Details** ページには、実行される Pipeline が表示されます。Pipeline が開始すると、各 Task 内の Task および Step が実行されます。以下を実行することができます。

   - 各 Step の実行にかかった時間を表示するには、Task にカーソルを合わせます。

   - Task をクリックし、Task の各 Step のログを表示します。

   - **Logs** タブをクリックして、Task の実行シーケンスに基づいてログを表示し、**Download** ボタンを使用してログをテキストファイルにダウンロードします。

図5.4 Pipeline Run



6. **From Git** オプションを使用して作成される Pipeline の場合、**Topology** ビューを使用して、開始後の Pipeline と対話することができます。

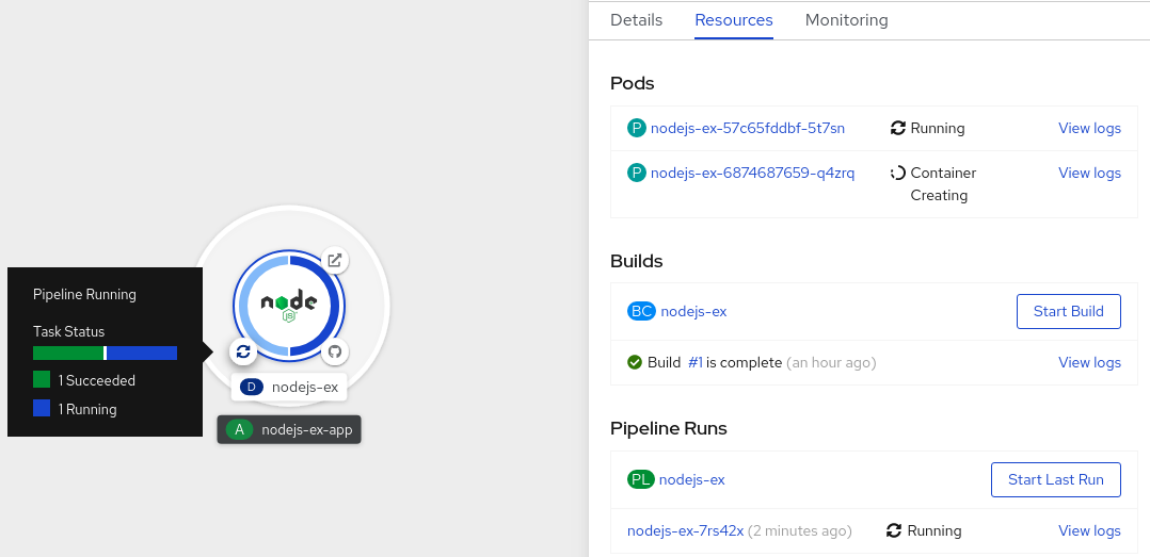
注記

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

a. 左側のナビゲーションパネルで **Topology** をクリックし、アプリケーションをクリックしてサイドパネルで Pipeline Run を表示します。

b. **Pipeline Runs** セクションで、**Start Last Run** をクリックし、直前のパラメーターおよびリソースと同じパラメーターおよびリソースを使用して新規 Pipeline Run を開始します。このオプションは、Pipeline Run が開始されていない場合は無効になります。

図5.5 Topology ビューの Pipeline



c. **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、アプリケーションの Pipeline Run のステータスを確認します。

5.5. 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、クラスターの Pipeline 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. **Pipeline Builder** ページで以下を行います。
   - 追加の Task、パラメーター、またはリソースを Pipeline に追加できます。
   - 変更する必要のある Task をクリックし、サイドパネルで Task の詳細を表示し、表示名、パラメーターおよびリソースなどの必要な Task の詳細を変更できます。
   - 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。

5.6. 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** 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。

第6章 RED HAT OPENSHIFT PIPELINES リリースノート

Red Hat OpenShift Pipelines は、以下を提供する Tekton プロジェクトをベースとするクラウドネイティブの CI/CD エクスペリエンスです。

- 標準の Kubernetes ネイティブパイプライン定義 (CRD)
- CI サーバー管理のオーバーヘッドのないサーバーレスのパイプライン。
- S2I、Buildah、JIB、Kaniko などの Kubernetes ツールを使用してイメージをビルドするための拡張性。
- Kubernetes ディストリビューションでの移植性。
- パイプラインと対話するための強力な CLI。
- OpenShift Container Platform Web コンソールの Developer パースペクティブと統合されたユーザーエクスペリエンス。

Red Hat OpenShift Pipeline の概要については、「OpenShift Pipeline について」を参照してください。

6.1. サポート

本書で説明されている手順に関連して問題が生じた場合には、 Red Hat カスタマーポータルにアクセスして、Red Hat テクノロジープレビュー機能のサポート範囲 について確認してください。

質問やフィードバックについては、製品チームに pipelines-interest@redhat.com 宛のメールを送信してください。

6.2. RED HAT OPENSHIFT PIPELINE テクノロジープレビュー 1.1 のリリースノート

6.2.1. 新機能

Red Hat OpenShift Pipelines テクノロジープレビュー (TP) 1.1 が OpenShift Container Platform 4.5 で利用可能になりました。Red Hat OpenShift Pipelines TP 1.1 が以下をサポートするように更新されています。

- Tekton Pipelines 0.14.3
- Tekton **tkn** CLI 0.11.0
- Tekton Triggers 0.6.1
- Tekton Catalog 0.14 をベースとする ClusterTask

以下では、修正および安定性の面での改善点に加え、OpenShift Pipeline 1.1 の主な新機能について説明します。

6.2.1.1. Pipeline

- Workspace を PipelineResource の代わりに使用できるようになりました。PipelineResource はデバッグが容易ではなく、スコープの制限があり、Task を再利用可能にしないため、

OpenShift Pipelines では Workspace を使用することが推奨されます。Workspace の詳細は、OpenShift Pipelines について参照してください。

- VolumeClaimTemplate の Workspace サポートが追加されました。
  - PipelineRun および TaskRun の VolumeClaimTemplate が Workspace のボリュームソースとして追加できるようになりました。次に、tekton-controller は Pipeline のすべての TaskRun の PVC として表示されるテンプレートを使用して PersistentVolumeClaim (PVC) を作成します。したがって、複数のタスクにまたがる Workspce をバインドするたびに PVC 設定を定義する必要はありません。
  - VolumeClaimTemplate がボリュームソースとして使用される場合の PersistentVolumeClaim の名前検索のサポートが、変数の置換を使用して利用できるようになりました。
- 監査を強化するサポート:
  - **PipelineRun.Status** フィールドには、Pipeline のすべての TaskRun のステータスと、PipelineRun の進捗をモニターするために PipelineRun をインスタンス化する際に使用する Pipeline 仕様が含まれるようになりました。
  - Pipeline の結果が Pipeline 仕様および **PipelineRun** ステータスに追加されました。
  - **TaskRun.Status** フィールドには、**TaskRun** のインスタンス化に使用される実際の Task 仕様が含まれるようになりました。
- デフォルトパラメーターを Condition に適用するサポート。
- ClusterTask を参照して作成される TaskRun は、**tekton.dev/task** ラベルではなく **tekton.dev/clusterTask** ラベルを追加するようになりました。
- **kubeconfigwriter** は、kubeconfig-creator Task でパイプラインリソースタイプクラスターの置き換えを有効にするために **ClientKeyData** および **ClientCertificateData** 設定を Resource 構造に追加できるようになりました。
- **feature-flags** および **config-defaults** ConfigMap の名前はカスタマイズ可能になりました。
- TaskRun で使用される PodTemplate の HostNetwork のサポートが利用可能になりました。
- Affinity Assistant が、Workspace ボリュームを共有する TaskRun のノードのアフィニティーをサポートするようになりました。デフォルトで、これは OpenShift Pipelines で無効にされます。
- PodTemplate は、Pod の起動時にコンテナーイメージのプルを許可するためにコンテナーランタイムが使用するシークレットを特定するために **imagePullSecrets** を指定するように更新されました。
- コントローラーが TaskRun の更新に失敗した場合に TaskRun コントローラーから警告イベントを出すためのサポート。
- アプリケーションまたはコンポーネントに属するリソースを特定するために、すべてのリソースに標準または推奨される k8s ラベルが追加されました。
- Entrypoint プロセスがシグナルについて通知されるようになり、これらのシグナルは Entrypoint プロセスの専用の PID グループを使用して伝播されるようになりました。
- PodTemplate は **TaskRunSpecs** を使用してランタイム時に Task レベルで設定できるようになりました。

- Kubernetes イベントを生成するサポート。
  - コントローラーは、追加の TaskRun ライフサイクルイベント (**taskrun started** および **taskrun running**) のイベントを生成するようになりました。
  - PipelineRun コントローラーは、Pipeline の起動時に毎回イベントを生成するようになりました。
- デフォルトの Kubernetes イベントのほかに、TaskRun の CloudEvents のサポートが利用可能になりました。コントローラーは、クラウドイベントとして create、started、および failed などの TaskRun イベントを送信するように設定できます。
- PipelineRun および TaskRun の場合に適切な名前を参照するための **$context.<taskRun|pipeline|pipelineRun>.name** 変数を使用するサポート。
- PipelineRun パラメーターの検証が、Pipeline で必要なすべてのパラメーターが PipelineRun によって提供できるようにするために利用可能になりました。これにより、PipelineRun は必要なパラメーターに加えて追加のパラメーターを指定することもできます。
- Pipeline YAML ファイルの **finally** フィールドを使用して、すべてのタスクが正常に終了するか、または Pipeline の Task の失敗後、Pipeline が終了する前に常に実行される Pipeline 内で Task を指定できるようになりました。
- **git-clone** ClusterTask が利用できるようになりました。

6.2.1.2. Pipelines CLI

- 組み込まれた Trigger バインディングのサポートが、**tkn evenlistener describe** コマンドで利用できるようになりました。
- 正しくないサブコマンドが使用される場合にサブコマンドを推奨し、提案するためのサポート。
- **tkn task describe** コマンドは、1 つのタスクのみが Pipeline に存在する場合にタスクを自動的に選択できるようになりました。
- **--use-param-defaults** フラグを **tkn task start** コマンドに指定することにより、デフォルトのパラメーター値を使用して Task を起動できるようになりました。
- **--workspace** オプションを **tkn pipeline start** または **tkn task start** コマンドで使用して、PipelineRun または TaskRun の volumeClaimTemplate を指定できるようになりました。
- **tkn pipelinerun logs** コマンドに、**finally** セクションに一覧表示される最終タスクのログが表示されるようになりました。
- インタラクティブモードのサポートが、以下の tkn リソース向けに **tkn task start** コマンドおよび **describe** サブコマンドに追加されました: **pipeline**、**pipelinerun**、**task**、**taskrun**、**clustertask**、および **pipelineresource**。
- **tkn version** コマンドで、クラスターにインストールされている Trigger のバージョンが表示されるようになりました。
- **tkn pipeline describe** コマンドで、Pipeline で使用される Task に指定されたパラメーター値およびタイムアウトが表示されるようになりました。
- 最近の PipelineRun または TaskRun をそれぞれ記述できるように、**tkn pipelinerun describe** および **tkn taskrun describe** コマンドの **--last** オプションのサポートが追加されました。

- **tkn pipeline describe** コマンドに、Pipeline の Task に適用される Condition が表示されるようになりました。
- **--no-headers** および **--all-namespaces** フラグを **tkn resource list** コマンドで使用できるようになりました。

6.2.1.3. トリガー

- 以下の Common Expression Language (CEL) 機能が利用できるようになりました。
  - **parseURL**: URL の一部を解析し、抽出します。
  - **parseJSON**: **deployment** webhook の **payload** フィールドの文字列に埋め込まれた JSON 値タイプを解析します。
- Bitbucket からの Webhook の新規インターセプターが追加されました。
- EventListener は、**kubectl get** コマンドで一覧表示される際の追加フィールドとして **Address URL** および **Available status** を表示します。
- TriggerTemplate パラメーターは、**$(params.<paramName>)** ではなく **$(tt.params.<paramName>)** 構文を使用するようになり、TriggerTemplate と ResourceTemplates パラメーター間で生じる混乱が軽減されました。
- EventListener CRD に **tolerations** を追加し、セキュリティーや管理上の問題によりすべてのノードにテイントのマークが付けられる場合でも EventListener が同じ設定でデプロイされるようにできるようになりました。
- EventListener Deployment の Readiness Probe を **URL/live** に追加できるようになりました。
- EventListener Trigger での TriggerBinding 仕様の埋め込みのサポート。
- Trigger リソースに推奨される **app.kubernetes.io** ラベルでアノテーションが付けられるようになりました。

6.2.2. 非推奨の機能

本リリースでは、以下の項目が非推奨になりました。

- **clustertask** コマンドおよび **clustertriggerbinding** コマンドを含む、クラスター全体のすべてのコマンドの **--namespace** または **-n** フラグが非推奨になりました。これは今後のリリースで削除されます。
- **ref** フィールドが優先されるため、EventListener 内の **triggers.bindings** の **name** フィールドは非推奨となり、今後のリリースで削除されます。
- **$(tt.params)** が優先されるため、**$(params)** を使用した TriggerTemplates の変数の補間が非推奨となり、これにより、Pipeline 変数の補間構文に関連した混乱が軽減されました。**$(params.<paramName>)** 構文は今後のリリースで削除されます。
- **tekton.dev/task** ラベルは ClusterTasks で非推奨になりました。
- **TaskRun.Status.ResourceResults.ResourceRef** フィールドは非推奨となり、今後削除されます。
- **tkn pipeline create**、 **tkn task create**、および **tkn resource create -f** サブコマンドが削除されました。

- namespace の検証が **tkn** コマンドから削除されました。
- **tkn ct start** コマンドのデフォルトタイムアウトの **1h** および **-t** フラグが削除されました。
- **s2i** ClusterTask が非推奨になりました。

6.2.3. 既知の問題

- Condition では Workspace はサポートされません。
- **--workspace** オプションとおよびインタラクティブモードは **tkn clustertask start** コマンドではサポートされていません。
- **$(params.<paramName>)** の後方互換性のサポートにより、TriggerTemplates がパイプライン固有のパラメーターで強制的に使用されます。 Triggers webhook が Trigger パラメーターとパイプラインパラメーターを区別できないためです。
- Pipeline メトリクスは、**tekton_taskrun_count** および **tekton_taskrun_duration_seconds_count** の promQL を実行する際に正しくない値を報告します。
- PipelineRun および TaskRun は、存在しない PVC 名が Workspace に指定されている場合でも、それぞれ **Running** および **Running(Pending)** の状態のままになります。

6.2.4. 修正された問題

- 以前のバージョンでは、Task および ClusterTask の名前が同じ場合、**tkn task delete <name> --trs** コマンドは、Task と ClusterTask の両方を削除しました。今回の修正により、コマンドは Task **<name>** で作成される TaskRun のみを削除するようになりました。
- 以前のバージョンでは、**tkn pr delete -p <name> --keep 2** コマンドは、**--keep** フラグと共に使用する場合に **-p** フラグを無視し、最新の 2 つの PipelineRun を除きすべての PipelineRun を削除しました。今回の修正により、コマンドは最新の 2 つの PipelineRun を除き、Pipeline **<name>** で作成される PipelineRun のみを削除するようになりました。
- **tkn triggertemplate describe** 出力には、YAML 形式ではなくテーブル形式で ResourceTemplates が表示されるようになりました。
- 以前のバージョンでは、**buildah** ClusterTask は、新規ユーザーがコンテナーに追加されると失敗していました。今回の修正により、この問題は解決されています。

6.3. RED HAT OPENSHIFT PIPELINE テクノロジープレビュー 1.0 のリリースノート

6.3.1. 新機能

Red Hat OpenShift Pipeline テクノロジープレビュー (TP) 1.0 が OpenShift Container Platform 4.5 で利用可能になりました。Red Hat OpenShift Pipeline TP 1.0 が以下をサポートするように更新されています。

- Tekton Pipelines 0.11.3
- Tekton **tkn** CLI 0.9.0
- Tekton Triggers 0.4.0

- Tekton Catalog 0.11 をベースとする ClusterTask

以下では、修正および安定性の面での改善点に加え、OpenShift Pipeline 1.0 の主な新機能について説明します。

6.3.1.1. Pipeline

- v1beta1 API バージョンのサポート。
- 改善された LimitRange のサポート。以前のバージョンでは、LimitRange は TaskRun および PipelineRun に対してのみ指定されていました。LimitRange を明示的に指定する必要がなくなりました。namespace 間で最小の LimitRange が使用されます。
- TaskResults および TaskParams を使用して Task 間でデータを共有するためのサポート。
- パイプラインは、**HOME** 環境変数および Step の **workingDir** を上書きしないように設定できるようになりました。
- Task Step と同様に、**sidecars** がスクリプトモードをサポートするようになりました。
- TaskRun **podTemplate** に別のスケジューラーの名前を指定できるようになりました。
- Star Array Notation を使用した変数置換のサポート。
- Tekton Controller は、個別の namespace を監視するように設定できるようになりました。
- Pipeline、Task、ClusterTask、Resource、および Condition の仕様に新規の説明フィールドが追加されました。
- Git PipelineResource へのプロキシーパラメーターの追加。

6.3.1.2. Pipelines CLI

- **describe** サブコマンドが以下の **tkn** リソースについて追加されました。**eventlistener**、**condition**、**triggertemplate**、**clustertask**、および **triggerbinding**。
- **v1beta1** についてのサポートが、**v1alpha1** の後方互換性と共に以下のコマンドに追加されました。**clustertask**、**task**、 **pipeline**、**pipelinerun**、および **taskrun**。
- 以下のコマンドで、**--all-namespaces** フラグオプションを使用してすべての namespace からの出力を一覧表示できるようになりました。
  - **tkn task list**
  - **tkn pipeline list**
  - **tkn taskrun list**
  - **tkn pipelinerun list**
    これらのコマンドの出力は、**--no-headers** フラグオプションを使用してヘッダーなしで情報を表示するように強化されています。
- **--use-param-defaults** フラグを **tkn pipelines start** コマンドに指定することにより、デフォルトのパラメーター値を使用して Pipeline を起動できるようになりました。
- Workspace のサポートが **tkn pipeline start** および **tkn task start** コマンドに追加されるようになりました。

- 新規の **clustertriggerbinding** コマンドが以下のサブコマンドと共に追加されました。**describe**、**delete**、および **list**。

- ローカルまたはリモートの **yaml** ファイルを使用してパイプラインの実行を直接開始できるようになりました。

- **describe** サブコマンドには、強化され、詳細化した出力が表示されるようになりました。**description**、**timeout**、 **param description**、および **sidecar status** などの新規フィールドの追加により、コマンドの出力に特定の **tkn** リソースについてのより詳細な情報が提供されるようになりました。

- **tkn task log** コマンドには、1 つのタスクが namespace に存在する場合にログが直接表示されるようになりました。

6.3.1.3. トリガー

- Trigger は **v1alpha1** および **v1beta1** の両方の Pipeline リソースを作成できるようになりました。

- 新規 Common Expression Language (CEL) インターセプター機能 **compareSecret** のサポート。この機能は、文字列と CEL 式のシークレットを安全な方法で比較します。

- EventListener Trigger レベルでの認証および認可のサポート。

6.3.2. 非推奨の機能

本リリースでは、以下の項目が非推奨になりました。

- 環境変数 **$HOME**、および Step 仕様の変数 **workingDir** が非推奨となり、今後のリリースで変更される可能性があります。現時点で Step コンテナーでは、**HOME** および**workingDir** が **/tekton/home** および **/workspace** にそれぞれ上書きされます。
  今後のリリースでは、これらの 2 つのフィールドは変更されず、コンテナーイメージおよび Task YAML で定義される値に設定されます。本リリースでは、フラグ **disable-home-env-overwrite** および **disable-working-directory-overwrite** を使用して、**HOME** および **workingDir** 変数の上書きを無効にします。

- 以下のコマンドは非推奨となり、今後のリリースで削除される可能性があります。

  - **tkn pipeline create**

  - **tkn task create**

- **tkn resource create** コマンドの **-f** フラグは非推奨になりました。これは今後のリリースで削除される可能性があります。

- **tkn clustertask create** コマンドの **-t** フラグおよび **--timeout** フラグ (秒単位の形式) は非推奨になりました。期間タイムアウトの形式のみがサポートされるようになりました (例: **1h30s**)。これらの非推奨のフラグは今後のリリースで削除される可能性があります。

6.3.3. 既知の問題

- 以前のバージョンの Red Hat OpenShift Pipelines からアップグレードする場合は、既存のデプロイメントを削除してから Red Hat OpenShift Pipelines バージョン 1.0 にアップグレードする必要があります。既存のデプロイメントを削除するには、まずカスタムリソースを削除してから Red Hat OpenShift Pipelines Operator をアンインストールする必要があります。詳細は、Red Hat OpenShift Pipelines のアンインストールについてのセクションを参照してください。

- 同じ **v1alpha1** Task を複数回送信すると、エラーが発生します。**v1alpha1** Task の再送信時に、**oc apply** ではなく **oc replace** を使用します。

- **buildah** ClusterTask は、新規ユーザーがコンテナーに追加されると機能しません。Operator がインストールされると、**buildah** ClusterTask の **--storage-driver** フラグが指定されていないため、フラグはデフォルト値に設定されます。これにより、ストレージドライバーが正しく設定されなくなることがあります。新規ユーザーが追加されると、storage-driver が間違っている場合に、**buildah** ClusterTask が以下のエラーを出して失敗します。

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

回避策として、**buildah-task.yaml** ファイルで **--storage-driver** フラグの値を **overlay** に手動で設定します。

1. **cluster-admin** として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. **oc edit** コマンドを使用して **buildah** ClusterTask を編集します。

```
$ oc edit clustertask buildah
```

buildah clustertask YAML ファイルの現行バージョンが **EDITOR** 環境変数で設定されたエディターで開かれます。

3. **steps** フィールドで、以下の **command** フィールドを見つけます。

```
 command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-
verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t',
'$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

4. **command** フィールドを以下に置き換えます。

```
 command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--
tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t',
'$(params.IMAGE)', '$(params.CONTEXT)']
```

5. ファイルを保存して終了します。

または、**Pipelines → Cluster Tasks → buildah** に移動して、**buildah** ClusterTask YAML ファイルを Web コンソール上で直接変更することもできます。**Actions** メニューから **Edit Cluster Task** を選択し、直前の手順のように **command** フィールドを置き換えます。

6.3.4. 修正された問題

- 以前のリリースでは、**DeploymentConfig** Task は、イメージのビルドがすでに進行中であっても新規デプロイメントビルドをトリガーしていました。これにより、Pipeline のデプロイメントが失敗していました。今回の修正により、**deploy task** コマンドが **oc rollout status** コマンドに置き換えられ、進行中のデプロイメントが終了するまで待機するようになりました。

- **APP_NAME** パラメーターのサポートが Pipeline テンプレートに追加されました。

- 以前のバージョンでは、Java S2I の Pipeline テンプレートはレジストリーでイメージを検索で

きませんでした。今回の修正により、イメージはユーザーによって提供される **IMAGE_NAME** パラメーターの代わりに既存イメージの PipelineResource を使用して検索されるようになりました。

- OpenShift Pipelines イメージはすべて、Red Hat Universal Base Images (UBI) をベースにしています。
- 以前のバージョンでは、Pipeline が **tekton-pipelines** 以外の namespace にインストールされている場合、**tkn version** コマンドは Pipeline のバージョンを **unknown** と表示していました。今回の修正により、**tkn version** コマンドにより、正しい Pipeline のバージョンがすべての namespace で表示されるようになりました。
- **-c** フラグは **tkn version** コマンドでサポートされなくなりました。
- 管理者以外のユーザーが ClusterTriggerBinding を一覧表示できるようになりました。
- EventListener CompareSecret 機能が、CEL インターセプターについて修正されました。
- **task** および **clustertask** の **list**、**describe**、および **start** サブコマンドは、Task および ClusterTask が同じ名前を持つ場合に出力に正常に表示されるようになりました。
- 以前のバージョンでは、OpenShift Pipelines Operator は特権付き SCC (Security Context Constraints) を変更していました。これにより、クラスターのアップグレード時にエラーが発生しました。このエラーは修正されています。
- **tekton-pipelines** namespace では、ConfigMap を使用して、すべての TaskRun および PipelineRun のタイムアウトが **default-timeout-minutes** フィールドの値に設定されるようになりました。
- 以前のバージョンでは、Web コンソールの Pipelines セクションは管理者以外のユーザーには表示されませんでした。この問題は解決されています。