OpenShift Container Platform 4.4

Pipeline

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

Last Updated: 2023-04-01

OpenShift Container Platform 4.4 Pipeline

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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

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。テクノロジープレビューの機能は、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ついての詳細は、https://access.redhat.com/support/offerings/techpreview/ を参照してください。

OpenShift Pipeline 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

- OpenShift Pipeline 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。
- OpenShift Pipeline 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。
- OpenShift Pipeline 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。
- OpenShift Pipeline 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. OPENSHIFT PIPELINE の概念

OpenShift Pipeline 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 のビルドを設定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設定されます。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 で設定されます。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。

PipelineResource

PipelineResource は、Pipeline Task の入力および出力として使用されるオブジェクトです。たとえば、入力が Git リポジトリーで、出力がその Git リポジトリーからビルドされるコンテナーイメージである場合、これらはどちらも PipelineResource として分類されます。PipelineResource は現在 Git リソース、イメージリソース、クラスターリソース、ストレージリソース、および CloudEvent リソースをサポートします。

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。

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

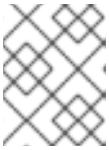
- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- **oc** CLI がインストールされていること。
- OpenShift Pipeline (**tkn**) CLI がローカルシステムにインストールされていること。

2.1. WEB コンソールでの 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 OpenShift Pipeline をインストールできます。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 **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. **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. **Create Operator Subscription** ページで以下を実行します。
   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、 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、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. **Subscribe** 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、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 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 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
```

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。

その他のリソース

- 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

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. 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. OpenShift Pipelines Operator をアンインストールします。

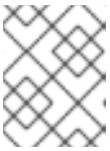
Operator のみをアンインストールしても、Operator のインストール時にデフォルトで作成される OpenShift Pipeline コンポーネントは削除されません。

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

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、 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 OpenShift Pipeline 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. 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 **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 PIPELINE を使用したアプリケーションの作成

OpenShift Pipeline 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。
- Pipeline および PipelineResources を作成し、アプリケーションの配信 Pipeline を定義します。
- 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、以下で設定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イス **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イス **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

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 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いる。
- リポジトリーへの管理者アクセスがある。

4.1. 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。
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
```

4.2. TASK について

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 で設定されます。Step 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、設定マップ、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。

Task は Git リソースなどの **inputs** パラメーター、およびレジストリーのイメージなどの **outputs** パラメーターを使用して他の Task と対話します。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。

以下は、Maven ベースの Java アプリケーションをビルドするための単一 Step の Maven Task の例です。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-build
spec:
  resources:
    inputs:
    - name: workspace-git
      targetPath: /
      type: git
  steps:
  - name: build
    image: maven:3.6.0-jdk-8-slim
    command:
    - /usr/bin/mvn
    args:
    - install
```

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。

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-1/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-1/01_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。

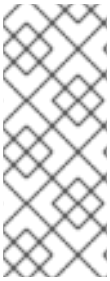
```
NAME                      DESCRIPTION   AGE
buildah                                 1 day ago
buildah-v0-11-3                         1 day ago
git-clone                               1 day ago
jib-maven                               1 day ago
kn                                      1 day ago
maven                                   1 day ago
openshift-client                        1 day ago
openshift-client-v0-11-3                1 day ago
s2i                                     1 day ago
s2i-dotnet-3                            1 day ago
s2i-dotnet-3-v0-11-3                    1 day ago
s2i-go                                  1 day ago
s2i-go-v0-11-3                          1 day ago
s2i-java-11                             1 day ago
s2i-java-11-v0-11-3                     1 day ago
s2i-java-8                              1 day ago
s2i-java-8-v0-11-3                      1 day ago
```

```
s2i-nodejs                 1 day ago
s2i-nodejs-v0-11-3         1 day ago
s2i-perl                   1 day ago
s2i-perl-v0-11-3           1 day ago
s2i-php                    1 day ago
s2i-php-v0-11-3            1 day ago
s2i-python-3               1 day ago
s2i-python-3-v0-11-3       1 day ago
s2i-ruby                   1 day ago
s2i-ruby-v0-11-3           1 day ago
s2i-v0-11-3                1 day ago
tkn                        1 day ago
```

4.4. PIPELINERESOURCES の定義および作成

PipelineResources は、Task の入力または出力として使用されるアーティファクトです。

Task の作成後に、実行時に Pipeline で使用される Git リポジトリーおよびイメージレジストリーの詳細が含まれる PipelineResources を作成します。



注記

pipelines-tutorial namespace ではなく別の namespace を使用している場合は、以下の手順のように使用している namespace でフロントエンドおよびバックエンドのイメージリソースを正しい URL に対して更新してください。以下に例を示します。

```
image-registry.openshift-image-registry.svc:5000/<namespace-name>/vote-api:latest
```

手順

1. フロントエンドアプリケーションの Git リポジトリーを定義する PipelineResource を作成します。

```
$ tkn resource create
? Enter a name for a pipeline resource : ui-repo
? Select a resource type to create : git
? Enter a value for url : http://github.com/openshift-pipelines/vote-ui.git
? Enter a value for revision : release-tech-preview-1
```

この出力では、**ui-repo** PipelineResource が作成されていることを検証します。

```
New git resource "ui-repo" has been created
```

2. フロントエンドイメージのプッシュ先に対して OpenShift Container Platform 内部イメージレジストリーを定義する PipelineResource を作成します。

```
$ tkn resource create
? Enter a name for a pipeline resource : ui-image
? Select a resource type to create : image
? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/ui:latest
? Enter a value for digest :
```

この出力では、**ui-image** PipelineResource が作成されていることを検証します。

```
New image resource "ui-image" has been created
```

3. バックエンドアプリケーションの Git リポジトリーを定義する PipelineResource を作成します。

```
$ tkn resource create
? Enter a name for a pipeline resource : api-repo
? Select a resource type to create : git
? Enter a value for url : http://github.com/openshift-pipelines/vote-api.git
? Enter a value for revision : release-tech-preview-1
```

出力では、**api-repo** PipelineResource が作成されていることを検証します。

```
New git resource "api-repo" has been created
```

4. バックエンドイメージのプッシュ先に対して OpenShift Container Platform 内部イメージレジストリーを定義する PipelineResource を作成します。

```
$ tkn resource create
? Enter a name for a pipeline resource : api-image
? Select a resource type to create : image
? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
? Enter a value for digest :
```

この出力では、**api-image** PipelineResource が作成されていることを検証します。

```
New image resource "api-image" has been created
```

5. 作成された **resources** の一覧を表示します。

```
$ tkn resource list
```

出力には、作成されているすべての PipelineResource が一覧表示されます。

```
NAME        TYPE    DETAILS
api-repo    git     url: http://github.com/openshift-pipelines/vote-api.git
ui-repo     git     url: http://github.com/openshift-pipelines/vote-ui.git
api-image   image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
ui-image    image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/ui:latest
```

4.5. 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、**inputs**、**outputs**、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、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。

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 リポジトリー **api-repo** および **ui-repo** 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 **api-image** および **ui-image** をビルドします。
- **api-image** および **ui-image** 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:
  resources:
  - name: git-repo
    type: git
  - name: image
    type: image
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  tasks:
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    resources:
      inputs:
      - name: source
        resource: git-repo
      outputs:
      - name: image
        resource: image
    params:
    - name: TLSVERIFY
      value: "false"
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    resources:
      inputs:
      - name: source
```

```
      resource: git-repo
  runAfter:
  - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  resources:
    inputs:
    - name: image
      resource: image
  params:
  - name: deployment
    value: $(params.deployment-name)
  runAfter:
  - apply-manifests
```

Pipeline 定義は、Pipeline の実行時に使用される 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。

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-1/01_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   ---        ---       ---        ---
```

4.6. 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 -r git-repo=api-repo -r image=api-image -p deployment-name=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 -r git-repo=ui-repo -r image=ui-image -p deployment-name=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 について

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 OpenShift Pipeline 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 システムを設定するように連携する以下の主なコンポーネントで設定されています。

- **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。

4.8. TRIGGER の PIPELINE への追加

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-1/03_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: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1alpha1
    kind: PipelineResource
    metadata:
      name: $(params.git-repo-name)-git-repo-$(uid)
    spec:
      type: git
      params:
      - name: revision
        value: $(params.git-revision)
      - name: url
        value: $(params.git-repo-url)

  - apiVersion: tekton.dev/v1alpha1
    kind: PipelineResource
    metadata:
      name: $(params.git-repo-name)-image-$(uid)
    spec:
      type: image
      params:
      - name: url
        value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(params.git-
repo-name):latest

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

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-1/03_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:
    - name: 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-1/03_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
```

4.9. 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 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。

4.10. 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-api** 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-api** のクローンを作成します。

```
$ git clone git@github.com:<your GitHub ID>/vote-api.git -b release-tech-preview-1
```

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-1
```

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。

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 OPENSHIFT PIPELINE 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、OpenShift Container Platform でアプリケーションを作成する間に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。

Pipeline の作成後に、デプロイした Pipeline を表示し、これとビジュアルに対話できます。

<Discrete><title>前提条件</title>

- OpenShift Container Platform クラスターにアクセスでき、 Web コンソール にログインしている。
- Operator をインストールするための管理者権限があり、 OpenShift Pipelines Operator をインストールしている。
- Developer パースペクティブ を使用している。
- プロジェクトを作成している。

</Discrete>

5.1. 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。
2. 必要な Pipeline をクリックし、**Pipeline Details** ページを表示します。これにより、Pipeline のすべての直列および並列の 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。
   - **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。

第6章 OPENSHIFT PIPELINE リリースノート

OpenShift Pipeline 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。

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 1.0 のリリースノート

6.2.1. 新機能

OpenShift Pipeline テクノロジープレビュー (TP) 1.0 が OpenShift Container Platform 4.4 で利用可能になりました。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。

6.2.1.1. 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。

6.2.1.2. 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。

6.2.1.3. 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。

6.2.2. 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。

- 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。

6.2.3. 既知の問題

- 以前のバージョンの OpenShift Pipeline 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 OpenShift Pipeline 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 OpenShift Pipelines Operator をアンインストールする必要があります。詳細は、OpenShift Pipeline 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。

6.2.4. 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。