OpenShift Container Platform 4.10

CI/CD

OpenShift Container Platform のビルド、パイプライン、および GitOps に関する情報

Last Updated: 2023-10-17

OpenShift Container Platform 4.10 CI/CD

OpenShift Container Platform のビルド、パイプライン、および GitOps に関する情報

法律上の通知

概要

OpenShift Container Platform 向けの CI/CD

目次

第1章 OPENSHIFT CONTAINER PLATFORM CI/CD の概要

OpenShift Container Platform は、開発者向けのエンタープライズ対応の Kubernetes プラットフォームであり、組織は継続的インテグレーション (CI) や継続的デリバリー (CD) などの DevOps プラクティスを通じてアプリケーションデリバリープロセスを自動化できます。組織のニーズを満たすために、OpenShift Container Platform は以下の CI/CD ソリューションを提供します。

- OpenShift Builds
- OpenShift Pipeline
- OpenShift GitOps

1.1. OPENSHIFT BUILDS

OpenShift Builds を使用すると、宣言型ビルドプロセスを使用してクラウドネイティブアプリを作成できます。BuildConfig オブジェクトの作成に使用する YAML ファイルでビルドプロセスを定義できます。この定義には、ビルドトリガー、入力パラメーター、ソースコードなどの属性が含まれます。デプロイされると、BuildConfig オブジェクトは通常、実行可能なイメージをビルドし、それをコンテナーイメージレジストリーにプッシュします。

OpenShift Builds は、ビルドストラテジーに対して以下の拡張可能なサポートを提供します。

- Docker ビルド
- Source-to-Image (S2I) ビルド
- カスタムビルド

詳細は、イメージビルドについて を参照してください。

1.2. OPENSHIFT PIPELINE

OpenShift Pipelines は、Kubernetes ネイティブの CI/CD フレームワークを提供して、CI/CD パイプラインの各ステップを独自のコンテナーで設計および実行します。独立して拡張し、予測可能な結果を伴うオンデマンドパイプラインに対応できます。

詳細は、OpenShift Pipelines について を参照してください。

1.3. OPENSHIFT GITOPS

OpenShift GitOps は、宣言型 GitOps エンジンとして Argo CD を使用するオペレーターです。これにより、マルチクラスター OpenShift および Kubernetes インフラストラクチャー全体で GitOps ワークフローが可能になります。管理者は、OpenShift GitOps を使用して、クラスターおよび開発ライフサイクル全体で Kubernetes ベースのインフラストラクチャーとアプリケーションを一貫して設定およびデプロイできます。

OpenShift GitOps について を参照してください。

1.4. JENKINS

Jenkins は、アプリケーションとプロジェクトの構築、テスト、およびデプロイのプロセスを自動化します。OpenShift Developer Tools は、OpenShift Container Platform と直接統合する Jenkins イメージを提供します。Jenkins は、Samples Operator テンプレートまたは認定 Helm チャートを使用して

OpenShift にデプロイできます。

第2章 ビルド

2.1. イメージビルドについて

2.1.1. ビルド

ビルドとは、入力パラメーターを結果として作成されるオブジェクトに変換するプロセスです。ほとんどの場合、このプロセスは入力パラメーターまたはソースコードを実行可能なイメージに変換するために使用されます。**BuildConfig** オブジェクトはビルドプロセス全体の定義です。

OpenShift Container Platform は、ビルドイメージからコンテナーを作成し、それらをコンテナーイメージレジストリーにプッシュして Kubernetes を使用します。

ビルドオブジェクトは共通の特性を共有します。これらには、ビルドの入力、ビルドプロセスの完了についての要件、ビルドプロセスのロギング、正常なビルドからのリリースのパブリッシュ、およびビルドの最終ステータスのパブリッシュが含まれます。ビルドはリソースの制限を利用し、CPU 使用、メモリー使用およびビルドまたは Pod の実行時間などのリソースの制限を指定します。

OpenShift Container Platform ビルドシステムは、ビルド API で指定される選択可能なタイプに基づくビルドストラテジーを幅広くサポートします。利用可能なビルドストラテジーは主に 3 つあります。

- Docker ビルド
- Source-to-Image (S2I) ビルド
- カスタムビルド

デフォルトで、docker ビルドおよび S2I ビルドがサポートされます。

ビルドの作成されるオブジェクトはこれを作成するために使用されるビルダーによって異なります。docker および S2I ビルドの場合、作成されるオブジェクトは実行可能なイメージです。カスタムビルドの場合、作成されるオブジェクトはビルダーイメージの作成者が指定するものになります。

さらに、パイプラインビルドストラテジーを使用して、高度なワークフローを実装することができます。

- 継続的インテグレーション
- 継続的デプロイメント

2.1.1.1. Docker ビルド

OpenShift Container Platform は Buildah を使用して Dockerfile からコンテナーイメージをビルドします。Dockerfile を使用したコンテナーイメージのビルドについての詳細は、Dockerfile リファレンスドキュメント を参照してください。

ヒント

buildArgs 配列を使用して Docker ビルド引数を設定する場合は、Dockerfile リファレンスドキュメントの ARG および FROM の対話方法 について参照してください。

2.1.1.2. Source-to-Image ビルド

Source-to-Image (S2I) は再現可能なコンテナーイメージをビルドするためのツールです。これはアプ

リケーションソースをコンテナーイメージに挿入し、新規イメージをアセンブルして実行可能なイメージを生成します。新規イメージはベースイメージ、ビルダーおよびビルドされたソースを組み込み、**buildah run** コマンドで使用することができます。S2I は増分ビルドをサポートします。これは以前にダウンロードされた依存関係や、以前にビルドされたアーティファクトなどを再利用します。

2.1.1.3. カスタムビルド

カスタムビルドストラテジーにより、開発者はビルドプロセス全体を対象とする特定のビルダーイメージを定義できます。独自のビルダーイメージを使用することにより、ビルドプロセスをカスタマイズできます。

カスタムビルダーイメージは、RPM またはベースイメージの構築など、ビルドプロセスのロジックに組み込まれるプレーンなコンテナーイメージです。

カスタムビルドは高いレベルの権限で実行されるため、デフォルトではユーザーが利用することはできません。クラスター管理者のパーミッションを持つ信頼できるユーザーのみにカスタムビルドを実行するためのアクセスが付与される必要があります。

2.1.1.4. パイプラインビルド

重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

開発者は、パイプラインビルドストラテジーを利用して Jenkins パイプラインプラグインで使用できるように Jenkins パイプラインを定義することができます。このビルドについては、他のビルドタイプの場合と同様に OpenShift Container Platform での起動、モニタリング、管理が可能です。

パイプラインワークフローは、ビルド設定に直接組み込むか、Git リポジトリーに配置してビルド設定で参照して **jenkinsfile** で定義します。

2.2. ビルド設定について

以下のセクションでは、ビルド、ビルド設定の概念を定義し、利用できる主なビルドストラテジーの概要を示します。

2.2.1. BuildConfig

ビルド設定は、単一のビルド定義と新規ビルドを作成するタイミングについてのトリガーセットを記述します。ビルド設定は **BuildConfig** で定義されます。 BuildConfig は、新規インスタンスを作成するために API サーバーへの POST で使用可能な REST オブジェクトのことです。

ビルド設定または **BuildConfig** は、ビルドストラテジーと 1 つまたは複数のソースを特徴としています。ストラテジーはプロセスを決定し、ソースは入力内容を提供します。

OpenShift Container Platform を使用したアプリケーションの作成方法の選択に応じて Web コンソールまたは CLI のいずれを使用している場合でも、**BuildConfig** は通常自動的に作成され、いつでも編集できます。**BuildConfig** を設定する部分や利用可能なオプションを理解しておくと、後に設定を手動で変

更する場合に役立ちます。

以下の **BuildConfig** の例では、コンテナーイメージのタグやソースコードが変更されるたびに新規ビルドが作成されます。

BuildConfig のオブジェクト定義

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" (1)
spec:
  runPolicy: "Serial" (2)
  triggers: (3)
    -
      type: "GitHub"
      github:
        secret: "secret101"
    - type: "Generic"
      generic:
        secret: "secret101"
    -
      type: "ImageChange"
  source: (4)
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy: (5)
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "ruby-20-centos7:latest"
  output: (6)
    to:
      kind: "ImageStreamTag"
      name: "origin-ruby-sample:latest"
  postCommit: (7)
      script: "bundle exec rake test"
```

(1) この仕様は、**ruby-sample-build** という名前の新規の **BuildConfig** を作成します。

(2) **runPolicy** フィールドは、このビルド設定に基づいて作成されたビルドを同時に実行できるかどうかを制御します。デフォルトの値は **Serial** です。 これは新規ビルドが同時にではなく、順番に実行されることを意味します。

(3) 新規ビルドを作成するトリガーのリストを指定できます。

(4) **source** セクションでは、ビルドのソースを定義します。ソースの種類は入力の主なソースを決定し、**Git** (コードのリポジトリーの場所を参照)、**Dockerfile** (インラインの Dockerfile からビルド) または **Binary** (バイナリーペイロードを受け入れる) のいずれかとなっています。複数のソースを一度に指定できます。各ソースタイプの詳細については、ビルド入力の作成を参照してください。

(5) **strategy** セクションでは、ビルドの実行に使用するビルドストラテジーを記述します。ここでは **Source**、**Docker** または **Custom** ストラテジーを指定できます。上記の例では、Source-to-image (S2I) がアプリケーションのビルドに使用する **ruby-20-centos7** コンテナーイメージを使用します。

6 コンテナーイメージが正常にビルドされた後に、これは **output** セクションで記述されているリポジトリーにプッシュされます。

7 **postCommit** セクションは、オプションのビルドフック を定義します。

2.3. ビルド入力の作成

以下のセクションでは、ビルド入力の概要、ビルドの動作に使用するソースコンテンツを提供するための入力の使用方法、およびビルド環境の使用およびシークレットの作成方法について説明します。

2.3.1. ビルド入力

ビルド入力は、ビルドが動作するために必要なソースコンテンツを提供します。以下のビルド入力を使用して OpenShift Cotainer Platform でソースを提供します。以下に優先される順で記載します。

- インラインの Dockerfile 定義
- 既存イメージから抽出したコンテンツ
- Git リポジトリー
- バイナリー (ローカル) 入力
- 入力シークレット
- 外部アーティファクト

複数の異なる入力を単一のビルドにまとめることができます。インラインの Dockerfile が優先されるため、別の入力で指定される Dockerfile という名前の他のファイルは上書きされます。バイナリー (ローカル) 入力および Git リポジトリーは併用できません。

入力シークレットは、ビルド時に使用される特定のリソースや認証情報をビルドで生成される最終アプリケーションイメージで使用不可にする必要がある場合や、シークレットリソースで定義される値を使用する必要がある場合に役立ちます。外部アーティファクトは、他のビルド入力タイプのいずれとしても利用できない別のファイルをプルする場合に使用できます。

ビルドを実行すると、以下が行われます。

1. 作業ディレクトリーが作成され、すべての入力内容がその作業ディレクトリーに配置されます。たとえば、入力 Git リポジトリーのクローンはこの作業ディレクトリーに作成され、入力イメージから指定されたファイルはターゲットのパスを使用してこの作業ディレクトリーにコピーされます。
2. ビルドプロセスによりディレクトリーが **contextDir** に変更されます (定義されている場合)。
3. インライン Dockerfile がある場合は、現在のディレクトリーに書き込まれます。
4. 現在の作業ディレクトリーにある内容が Dockerfile、カスタムビルダーのロジック、または **assemble** スクリプトが参照するビルドプロセスに提供されます。つまり、ビルドでは **contextDir** 内にない入力コンテンツは無視されます。

以下のソース定義の例には、複数の入力タイプと、入力タイプの統合方法の説明が含まれています。それぞれの入力タイプの定義方法に関する詳細は、各入力タイプについての個別のセクションを参照してください。

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

(1) 作業ディレクトリーにクローンされるビルド用のリポジトリー

(2) **myinputimage** の **/usr/lib/somefile.jar** は、**<workingdir>/app/dir/injected/dir** に保存されます。

(3) ビルドの作業ディレクトリーは **<original_workingdir>/app/dir** になります。

(4) このコンテンツを含む Dockerfile は **<original_workingdir>/app/dir** に作成され、この名前が指定された既存ファイルは上書きされます。

2.3.2. Dockerfile ソース

dockerfile の値が指定されると、このフィールドの内容は、**dockerfile** という名前のファイルとしてディスクに書き込まれます。これは、他の入力ソースが処理された後に実行されるので、入力ソースリポジトリーのルートディレクトリーに Dockerfile が含まれる場合は、これはこの内容で上書きされます。

ソースの定義は **BuildConfig** の **spec** セクションに含まれます。

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

(1) **dockerfile** フィールドには、ビルドされるインライン Dockerfile が含まれます。

関連情報

- このフィールドは、通常は Dockerfile を docker ストラテジービルドに指定するために使用されます。

2.3.3. イメージソース

追加のファイルは、イメージを使用してビルドプロセスに渡すことができます。インプットイメージは **From** および **To** イメージターゲットが定義されるのと同じ方法で参照されます。つまり、コンテナーイメージとイメージストリームタグの両方を参照できます。イメージとの関連で、1 つまたは複数のパスのペアを指定して、ファイルまたはディレクトリーのパスを示し、イメージと宛先をコピーしてビルドコンテキストに配置する必要があります。

ソースパスは、指定したイメージ内の絶対パスで指定してください。宛先は、相対ディレクトリーパス

でなければなりません。ビルド時に、イメージは読み込まれ、指定のファイルおよびディレクトリーはビルドプロセスのコンテキストディレクトリーにコピーされます。これは、ソースリポジトリーのコンテンツのクローンが作成されるディレクトリーと同じです。ソースパスの末尾は /. であり、ディレクトリーのコンテンツがコピーされますが、ディレクトリー自体は宛先で作成されません。

イメージの入力は、**BuildConfig** の **source** の定義で指定します。

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

1. 1 つ以上のインプットイメージおよびファイルの配列
2. コピーされるファイルが含まれるイメージへの参照
3. ソース/宛先パスの配列
4. ビルドプロセスで対象のファイルにアクセス可能なビルドルートへの相対パス
5. 参照イメージの中からコピーするファイルの場所
6. 認証情報がインプットイメージにアクセスするのに必要な場合に提供されるオプションのシークレット

注記

クラスターが **ImageContentSourcePolicy** オブジェクトを使用してリポジトリーのミラーリングを設定する場合、ミラーリングされたレジストリーにグローバルプルシークレットのみを使用できます。プロジェクトにプルシークレットを追加することはできません。

オプションとして、インプットイメージにプルシークレットが必要な場合、プルシークレットをビルドによって使用されるサービスアカウントにリンクできます。デフォルトで、ビルドは **builder** サービスアカウントを使用します。シークレットにインプットイメージをホストするリポジトリーに一致する認証情報が含まれる場合、プルシークレットはビルドに自動的に追加されます。プルシークレットをビルドで使用されるサービスアカウントにリンクするには、以下を実行します。

-

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

注記

この機能は、カスタムストラテジーを使用するビルドについてサポートされません。

2.3.4. Git ソース

ソースコードは、指定されている場合は指定先の場所からフェッチされます。

インラインの Dockerfile を指定する場合は、これにより Git リポジトリーの **contextDir** 内にある Dockerfile が上書きされます。

ソースの定義は **BuildConfig** の **spec** セクションに含まれます。

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

1. **git** フィールドには、ソースコードのリモート Git リポジトリーへの URI (Uniform Resource Identifier) が含まれます。特定の Git リファレンスをチェックアウトするには、**ref** フィールドの値を指定する必要があります。SHA1 タグまたはブランチ名は、**ref** として有効です。**ref** フィールドのデフォルト値は **master** です。

2. **contextDir** フィールドでは、ビルドがアプリケーションのソースコードを検索する、ソースコードのリポジトリー内のデフォルトの場所を上書きできます。アプリケーションがサブディレクトリーに存在する場合には、このフィールドを使用してデフォルトの場所 (root フォルダー) を上書きすることができます。

3. オプションの **dockerfile** フィールドがある場合は、Dockerfile を含む文字列を指定してください。 この文字列は、ソースリポジトリーに存在する可能性のある Dockerfile を上書きします。

ref フィールドにプル要求が記載されている場合には、システムは **git fetch** 操作を使用して **FETCH_HEAD** をチェックアウトします。

ref の値が指定されていない場合は、OpenShift Container Platform はシャロークローン (**--depth=1**) を実行します。この場合、デフォルトのブランチ (通常は **master**) での最新のコミットに関連するファイルのみがダウンロードされます。これにより、リポジトリーのダウンロード時間が短縮されます (詳細のコミット履歴はありません)。指定リポジトリーのデフォルトのブランチで完全な **git clone** を実行するには、**ref** をデフォルトのブランチ名に設定します (例: **main**)。

警告

中間者 (MITM) TLS ハイジャックまたはプロキシーされた接続の再暗号化を実行するプロキシーを通過する Git クローンの操作は機能しません。

2.3.4.1. プロキシーの使用

プロキシーの使用によってのみ Git リポジトリーにアクセスできる場合は、使用するプロキシーをビルド設定の **source** セクションで定義できます。HTTP および HTTPS プロキシーの両方を設定できます。いずれのフィールドもオプションです。**NoProxy** フィールドで、プロキシーを実行しないドメインを指定することもできます。

注記

実際に機能させるには、ソース URI で HTTP または HTTPS プロトコルを使用する必要があります。

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

注記

パイプラインストラテジーのビルドの場合には、現在 Jenkins の Git プラグインに制約があるので、Git プラグインを使用する Git の操作では **BuildConfig** に定義された HTTP または HTTPS プロキシーは使用されません。Git プラグインは、Jenkins UI の Plugin Manager パネルで設定されたプロキシーのみを使用します。どのジョブであっても、Jenkins 内の Git のすべての対話にはこのプロキシーが使用されます。

関連情報

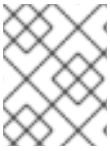
- Jenkins UI でのプロキシーの設定方法については、JenkinsBehindProxy を参照してください。

2.3.4.2. ソースクローンのシークレット

ビルダー Pod には、ビルドのソースとして定義された Git リポジトリーへのアクセスが必要です。ソースクローンのシークレットは、ビルダー Pod に対し、プライベートリポジトリーや自己署名証明書または信頼されていない SSL 証明書が設定されたリポジトリーなどの通常アクセスできないリポジトリーへのアクセスを提供するために使用されます。

以下は、サポートされているソースクローンのシークレット設定です。

- .gitconfig ファイル
- Basic 認証
- SSH キー認証
- 信頼されている認証局



注記

特定のニーズに対応するために、これらの設定の組み合わせを使用することもできます。

2.3.4.2.1. ソースクローンシークレットのビルド設定への自動追加

BuildConfig が作成されると、OpenShift Container Platform はソースクローンのシークレット参照を自動生成します。この動作により、追加の設定なしに、作成されるビルドが参照されるシークレットに保存された認証情報を自動的に使用できるようになり、リモート Git リポジトリーに対する認証が可能になります。

この機能を使用するには、Git リポジトリーの認証情報を含むシークレットが **BuildConfig** が後に作成される namespace になければなりません。このシークレットには、接頭辞 **build.openshift.io/source-secret-match-uri-** で開始するアノテーション 1 つ以上含まれている必要もあります。これらの各アノテーションの値には、以下で定義される URI (Uniform Resource Identifier) パターンを使用します。これは以下のように定義されます。ソースクローンのシークレット参照なしに **BuildConfig** が作成され、Git ソースの URI がシークレットのアノテーションの URI パターンと一致する場合に、OpenShift Container Platform はそのシークレットへの参照を **BuildConfig** に自動的に挿入します。

前提条件

URI パターンには以下を含める必要があります。

- 有効なスキーム: ***://**、 **git://**、**http://**、**https://** または **ssh://**
- ホスト: *` または有効なホスト名、あるいは ***.** が先頭に指定された IP アドレス
- パス: /* または、 / の後に * 文字などの文字がオプションで後に続きます。

上記のいずれの場合でも、* 文字はワイルドカードと見なされます。

重要

URI パターンは、RFC3986 に準拠する Git ソースの URI と一致する必要があります。URI パターンにユーザー名 (またはパスワード) のコンポーネントを含めないようにしてください。

たとえば、Git リポジトリーの URL に **ssh://git@bitbucket.atlassian.com:7999/ATLASSIAN jira.git** を使用する場合に、ソースのシークレットは、**ssh://bitbucket.atlassian.com:7999/*** として指定する必要があります (**ssh://git@bitbucket.atlassian.com:7999/*** ではありません)。

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

手順

複数のシークレットが特定の **BuildConfig** の Git URI と一致する場合は、OpenShift Container Platform は一致する文字列が一番長いシークレットを選択します。これは、以下の例のように基本的な上書きを許可します。

以下の部分的な例では、ソースクローンのシークレットの一部が 2 つ表示されています。1 つ目は、HTTPS がアクセスする **mycorp.com** ドメイン内のサーバーに一致しており、2 つ目は **mydev1.mycorp.com** および **mydev2.mycorp.com** のサーバーへのアクセスを上書きします。

```
kind: Secret
apiVersion: v1
metadata:
  name: matches-all-corporate-servers-https-only
  annotations:
```

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

- 以下のコマンドを使用して、**build.openshift.io/source-secret-match-uri-** アノテーションを既存のシークレットに追加します。

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

2.3.4.2.2. ソースクローンシークレットの手動による追加

ソースクローンのシークレットは、ビルド設定に手動で追加できます。 **sourceSecret** フィールドを **BuildConfig** 内の **source** セクションに追加してから、作成したシークレットの名前に設定して実行できます。この例では **basicsecret** です。

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

手順

oc set build-secret コマンドを使用して、既存のビルド設定にソースクローンのシークレットを設定することも可能です。

- 既存のビルド設定にソースクローンシークレットを設定するには、以下のコマンドを実行します。

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

2.3.4.2.3. .gitconfig ファイルからのシークレットの作成

アプリケーションのクローンが **.gitconfig** ファイルに依存する場合、そのファイルが含まれるシークレットを作成できます。これをビルダーサービスアカウントおよび **BuildConfig** に追加します。

手順

- **.gitconfig** ファイルからシークレットを作成するには、以下を実行します。

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

注記

.gitconfig ファイルの **http** セクションが **sslVerify=false** に設定されている場合は、SSL 検証をオフにすることができます。

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

2.3.4.2.4. セキュリティー保護された Git の .gitconfig ファイルからのシークレットの作成

Git サーバーが 2 方向の SSL、ユーザー名とパスワードでセキュリティー保護されている場合には、ソースビルドに証明書ファイルを追加して、**.gitconfig** ファイルに証明書ファイルへの参照を追加する必要があります。

前提条件

- Git 認証情報が必要です。

手順

ソースビルドに証明書ファイルを追加して、**.gitconfig** ファイルに証明書ファイルへの参照を追加します。

1. **client.crt**、**cacert.crt**、および **client.key** ファイルをアプリケーションソースコードの **/var/run/secrets/openshift.io/source/** フォルダーに追加します。
2. サーバーの **.gitconfig** ファイルに、以下のように **[http]** セクションを追加します。

```
# cat .gitconfig
```

出力例

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

3. シークレットを作成します。

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

1. ユーザーの Git ユーザー名
2. このユーザーのパスワード

重要

パスワードを再度入力しなくてもよいように、ビルドに Source-to-Image (S2I) イメージを指定するようにしてください。ただし、リポジトリーをクローンできない場合には、ビルドをプロモートするためにユーザー名とパスワードを指定する必要があります。

関連情報

- アプリケーションソースコードの **/var/run/secrets/openshift.io/source/** フォルダー。

2.3.4.2.5. ソースコードの基本的な認証からのシークレットの作成

Basic 認証では、SCM (software configuration management) サーバーに対して認証する場合に **--username** と **--password** の組み合わせ、またはトークンが必要です。

前提条件

- プライベートリポジトリーにアクセスするためのユーザー名およびパスワード。

手順

1. シークレットを先に作成してから、プライベートリポジトリーにアクセスするために **--username** および **--password** を使用してください。

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

2. トークンで Basic 認証のシークレットを作成します。

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

2.3.4.2.6. ソースコードの SSH キー認証からのシークレットの作成

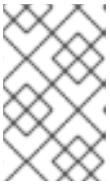
SSH キーベースの認証では、プライベート SSH キーが必要です。

リポジトリーのキーは通常 **$HOME/.ssh/** ディレクトリーにあり、デフォルトで **id_dsa.pub**、**id_ecdsa.pub**、**id_ed25519.pub**、または **id_rsa.pub** という名前が付けられています。

手順

1. SSH キーの認証情報を生成します。

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



注記

SSH キーのパスフレーズを作成すると、OpenShift Container Platform でビルドができなくなります。パスフレーズを求めるプロンプトが出されても、ブランクのままにします。

パブリックキーと、それに対応するプライベートキーのファイルが 2 つ作成されます (**id_dsa**、**id_ecdsa**、**id_ed25519** または **id_rsa** のいずれか)。これらが両方設定されたら、パブリックキーのアップロード方法についてソースコントロール管理 (SCM) システムのマニュアルを参照してください。プライベートキーは、プライベートリポジトリーにアクセスするために使用されます。

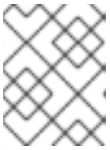
2. SSH キーを使用してプライベートリポジトリーにアクセスする前に、シークレットを作成します。

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

1 オプション: このフィールドを追加すると、厳密なサーバーホストキーチェックが有効になります。

警告

シークレットの作成中に **known_hosts** ファイルをスキップすると、ビルドが中間者 (MITM) 攻撃を受ける可能性があります。



注記

know_hosts ファイルにソースコードのホストのエントリーが含まれていることを確認してください。

2.3.4.2.7. ソースコードの信頼されている認証局からのシークレットの作成

Git clone の操作時に信頼される TLS (Transport Layer Security) 認証局 (CA) のセットは OpenShift Container Platform インフラストラクチャーイメージにビルドされます。Git サーバーが自己署名の証明書を使用するか、イメージで信頼されていない認証局によって署名された証明書を使用する場合には、その証明書が含まれるシークレットを作成するか、TLS 検証を無効にしてください。

CA 証明書のシークレットを作成した場合に、OpenShift Container Platform はその証明書を使用して、Git clone 操作時に Git サーバーにアクセスします。存在する TLS 証明書をどれでも受け入れてしまう Git の SSL 検証の無効化に比べ、この方法を使用するとセキュリティーレベルが高くなります。

手順

CA 証明書ファイルでシークレットを作成します。

1. CA が中間認証局を使用する場合には、**ca.crt** ファイルにすべての CA の証明書を統合します。以下のコマンドを入力します。

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

   a. シークレットを作成します。

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

   1 **ca.crt** というキーの名前を使用する必要があります。

2.3.4.2.8. ソースシークレットの組み合わせ

特定のニーズに対応するために上記の方法を組み合わせてソースクローンのシークレットを作成することができます。

2.3.4.2.8.1. .gitconfig ファイルでの SSH ベースの認証シークレットの作成

SSH ベースの認証シークレットと **.gitconfig** ファイルなど、特定のニーズに応じてソースクローンシークレットを作成するための複数の異なる方法を組み合わせることができます。

前提条件

- SSH 認証
- .gitconfig ファイル

手順

- **.gitconfig** ファイルを使用して SSH ベースの認証シークレットを作成するには、以下を実行します。

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

2.3.4.2.8.2. .gitconfig ファイルと CA 証明書を組み合わせるシークレットの作成

.gitconfig ファイルおよび認証局 (CA) 証明書を組み合わせるシークレットなど、特定のニーズに応じてソースクローンシークレットを作成するための複数の異なる方法を組み合わせることができます。

前提条件

- .gitconfig ファイル

- CA 証明書

手順

- **.gitconfig** ファイルと CA 証明書を組み合わせてシークレットを作成するには、以下を実行します。

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

2.3.4.2.8.3. CA 証明書ファイルを使用した Basic 認証のシークレットの作成

Basic 認証および CA (certificate authority) 証明書を組み合わせるシークレットなど、特定のニーズに応じてソースクローンシークレットを作成するための複数の異なる方法を組み合わせることができます。

前提条件

- Basic 認証の認証情報
- CA 証明書

手順

- CA 証明書ファイルを使用して Basic 認証のシークレットを作成し、以下を実行します。

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

2.3.4.2.8.4. .gitconfig ファイルを使用した Basic 認証シークレットの作成

Basic 認証および **.gitconfig** ファイルを組み合わせるシークレットなど、特定のニーズに応じてソースクローンシークレットを作成するための複数の異なる方法を組み合わせることができます。

前提条件

- Basic 認証の認証情報
- **.gitconfig** ファイル

手順

- **.gitconfig** ファイルで Basic 認証のシークレットを作成するには、以下を実行します。

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

2.3.4.2.8.5. .gitconfig ファイルと CA 証明書を使用した Basic 認証シークレットの作成

Basic 認証、**.gitconfig** ファイルおよび CA 証明書を組み合わせるシークレットなど、特定のニーズに応じてソースクローンシークレットを作成するための複数の異なる方法を組み合わせることができます。

前提条件

- Basic 認証の認証情報
- **.gitconfig** ファイル
- CA 証明書

手順

- **.gitconfig** ファイルと CA 証明書ファイルを合わせて Basic 認証シークレットを作成するには、以下を実行します。

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

2.3.5. バイナリー (ローカル) ソース

ローカルのファイルシステムからビルダーにコンテンツをストリーミングすることは、**Binary** タイプのビルドと呼ばれています。このビルドについての **BuildConfig.spec.source.type** の対応する値は **Binary** です。

このソースタイプは、**oc start-build** のみをベースとして使用される点で独特なタイプです。

注記

バイナリータイプのビルドでは、ローカルファイルシステムからコンテンツをストリーミングする必要があります。そのため、バイナリータイプのビルドを自動的にトリガーすること (例: イメージの変更トリガーなど) はできません。これは、バイナリーファイルを提供することができないためです。同様に、Web コンソールからバイナリータイプのビルドを起動することはできません。

バイナリービルドを使用するには、以下のオプションのいずれかを指定して **oc start-build** を呼び出します。

- **--from-file**: 指定したファイルのコンテンツはバイナリーストリームとしてビルダーに送信されます。ファイルに URL を指定することもできます。次に、ビルダーはそのデータをビルドコンテキストの上に、同じ名前のファイルに保存します。
- **--from-dir** および **--from-repo**: コンテンツはアーカイブされて、バイナリーストリームとしてバイナリーに送信されます。次に、ビルダーはビルドコンテキストディレクトリー内にアーカイブのコンテンツをデプロイメントします。**--from-dir** を使用して、デプロイメントされるアーカイブに URL を指定することもできます。

- **--from-archive**: 指定したアーカイブはビルダーに送信され、ビルドコンテキストディレクトリーにデプロイメントされます。このオプションは **--from-dir** と同様に動作しますが、このオプションの引数がディレクトリーの場合には常にアーカイブがホストに最初に作成されます。

上記のそれぞれの例では、以下のようになります。

- **BuildConfig** に **Binary** のソースタイプが定義されている場合には、これは事実上無視され、クライアントが送信する内容に置き換えられます。

- **BuildConfig** に **Git** のソースタイプが定義されている場合には、**Binary** と **Git** は併用できないので、動的に無効にされます。 この場合、ビルダーに渡されるバイナリーストリームのデータが優先されます。

ファイル名ではなく、HTTP または HTTPS スキーマを使用する URL を **--from-file** や **--from-archive** に渡すことができます。**--from-file** で URL を指定すると、ビルダーイメージのファイル名は Web サーバーが送信する **Content-Disposition** ヘッダーか、ヘッダーがない場合には URL パスの最後のコンポーネントによって決定されます。認証形式はどれもサポートされておらず、カスタムの TLS 証明書を使用したり、証明書の検証を無効にしたりできません。

oc new-build --binary=true を使用すると、バイナリービルドに関連する制約が実施されるようになります。作成される **BuildConfig** のソースタイプは **Binary** になります。 つまり、この **BuildConfig** のビルドを実行するための唯一の有効な方法は、**--from** オプションのいずれかを指定して **oc start-build** を使用し、必須のバイナリーデータを提供する方法になります。

Dockerfile および **contextDir** のソースオプションは、バイナリービルドに関して特別な意味を持ちます。

Dockerfile はバイナリービルドソースと合わせて使用できます。Ddockerfile を使用し、バイナリーストリームがアーカイブの場合には、そのコンテンツはアーカイブにある Dockerfile の代わりとして機能します。Dockerfile が **--from-file** の引数と合わせて使用されている場合には、ファイルの引数は Dockerfile となり、Dockerfile の値はバイナリーストリームの値に置き換わります。

バイナリーストリームがデプロイメントされたアーカイブのコンテンツをカプセル化する場合には、**contextDir** フィールドの値はアーカイブ内のサブディレクトリーと見なされます。 有効な場合には、ビルド前にビルダーがサブディレクトリーに切り替わります。

2.3.6. 入力シークレットおよび設定マップ

重要

入力シークレットおよび設定マップのコンテンツがビルドの出力コンテナーイメージに表示されないようにするには、Docker build と source-to-image build ストラテジーでビルドボリュームを使用します。

シナリオによっては、ビルド操作で、依存するリソースにアクセスするための認証情報や他の設定データが必要になる場合がありますが、この情報をソースコントロールに配置するのは適切ではありません。この場合は、入力シークレットおよび入力設定マップを定義することができます。

たとえば、Maven を使用して Java アプリケーションをビルドする場合、プライベートキーを使用してアクセスされる Maven Central または JCenter のプライベートミラーをセットアップできます。そのプライベートミラーからライブラリーをダウンロードするには、以下を指定する必要があります。

1. ミラーの URL および接続の設定が含まれる **settings.xml** ファイル。

2. ~/**.ssh/id_rsa** などの、設定ファイルで参照されるプライベートキー。

セキュリティー上の理由により、認証情報はアプリケーションイメージで公開しないでください。

以下の例は Java アプリケーションについて説明していますが、**/etc/ssl/certs** ディレクトリー、API キーまたはトークン、ラインセンスファイルなどに SSL 証明書を追加する場合に同じ方法を使用できます。

2.3.6.1. シークレットの概要

Secret オブジェクトタイプはパスワード、OpenShift Container Platform クライアント設定ファイル、**dockercfg** ファイル、プライベートソースリポジトリーの認証情報などの機密情報を保持するメカニズムを提供します。シークレットは機密内容を Pod から切り離します。シークレットはボリュームプラグインを使用してコンテナーにマウントすることも、システムが Pod の代わりにシークレットを使用して各種アクションを実行することもできます。

YAML シークレットオブジェクト定義

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

(1) シークレットにキー名および値の構造を示しています。

(2) **data** フィールドでキーに使用できる形式は、Kubernetes identifiers glossary の **DNS_SUBDOMAIN** 値のガイドラインに従う必要があります。

(3) **data** マップのキーに関連付けられる値は base64 でエンコーディングされている必要があります。

(4) **stringData** マップのエントリーが base64 に変換され、このエントリーは自動的に **data** マップに移動します。このフィールドは書き込み専用です。値は **data** フィールドによってのみ返されます。

(5) **stringData** マップのキーに関連付けられた値は単純なテキスト文字列で設定されます。

2.3.6.1.1. シークレットのプロパティー

キーのプロパティーには以下が含まれます。

- シークレットデータはその定義とは別に参照できます。
- シークレットデータのボリュームは一時ファイルストレージ機能 (tmpfs) でサポートされ、ノードで保存されることはありません。
- シークレットデータは namespace 内で共有できます。

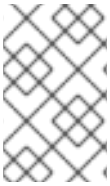
2.3.6.1.2. シークレットの種類

type フィールドの値で、シークレットのキー名と値の構造を指定します。このタイプを使用して、シークレットオブジェクトにユーザー名とキーの配置を実行できます。検証の必要がない場合には、デフォルト設定の **opaque** タイプを使用してください。

以下のタイプから 1 つ指定して、サーバー側で最小限の検証をトリガーし、シークレットデータに固有のキー名が存在することを確認します。

- **kubernetes.io/service-account-token**。サービスアカウントトークンを使用します。
- **kubernetes.io/dockercfg**.必須の Docker 認証には **.dockercfg** ファイルを使用します。
- **kubernetes.io/dockerconfigjson**.必須の Docker 認証には **.docker/config.json** ファイルを使用します。
- **kubernetes.io/basic-auth**.Basic 認証で使用します。
- **kubernetes.io/ssh-auth**.SSH キー認証で使用します。
- **kubernetes.io/tls**.TLS 認証局で使用します。

検証の必要がない場合には **type= Opaque** と指定します。これは、シークレットがキー名または値の規則に準拠しないという意味です。**opaque** シークレットでは、任意の値を含む、体系化されていない **key:value** ペアも利用できます。



注記

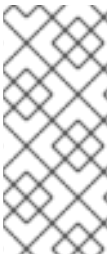
example.com/my-secret-type などの他の任意のタイプを指定できます。これらのタイプはサーバー側では実行されませんが、シークレットの作成者がその種類のキー/値の要件に従う意図があることを示します。

2.3.6.1.3. シークレットの更新

シークレットの値を変更する場合、すでに実行されている Pod で使用される値は動的に変更されません。シークレットを変更するには、元の Pod を削除してから新規の Pod を作成する必要があります (同じ **PodSpec** を使用する場合があります)。

シークレットの更新は、新規コンテナーイメージのデプロイと同じワークフローで実行されます。**kubectl rolling-update** コマンドを使用できます。

シークレットの **resourceVersion** 値は参照時に指定されません。したがって、シークレットが Pod の起動と同じタイミングで更新される場合、Pod に使用されるシークレットのバージョンは定義されません。



注記

現時点で、Pod の作成時に使用されるシークレットオブジェクトのリソースバージョンを確認することはできません。コントローラーが古い **resourceVersion** を使用して Pod を再起動できるように、Pod がこの情報を報告できるようにすることが予定されています。それまでは既存シークレットのデータを更新せずに別の名前で新規のシークレットを作成します。

2.3.6.2. シークレットの作成

シークレットに依存する Pod を作成する前に、シークレットを作成する必要があります。

シークレットの作成時に以下を実行します。

- シークレットデータでシークレットオブジェクトを作成します。
- Pod のサービスアカウントをシークレットの参照を許可するように更新します。
- シークレットを環境変数またはファイルとして使用する Pod を作成します (**secret** ボリュームを使用)。

手順

- 作成コマンドを使用して JSON または YAML ファイルのシークレットオブジェクトを作成できます。

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

たとえば、ローカルの **.docker/config.json** ファイルからシークレットを作成できます。

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

このコマンドにより、**dockerhub** という名前のシークレットの JSON 仕様が生成され、オブジェクトが作成されます。

YAML の不透明なシークレットオブジェクトの定義

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

1 **opaque** シークレットを指定します。

Docker 設定の JSON ファイルシークレットオブジェクトの定義

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

.dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXIzCg== 2
```

1 シークレットが docker 設定の JSON ファイルを使用することを指定します。

2 docker 設定 JSON ファイルを base64 でエンコードした出力

2.3.6.3. シークレットの使用

シークレットの作成後に、Pod を作成してシークレットを参照し、ログを取得し、Pod を削除することができます。

手順

1. シークレットを参照する Pod を作成します。

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

2. ログを取得します。

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

3. Pod を削除します。

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

関連情報

- シークレットデータを含む YAML ファイルのサンプル

4 つのファイルを作成する YAML シークレット

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

1. デコードされる値が含まれるファイル
2. デコードされる値が含まれるファイル
3. 提供される文字列が含まれるファイル
4. 提供されるデータが含まれるファイル

シークレットデータと共にボリュームのファイルが設定された Pod の YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
```

```
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
          # name must match the volume name below
          - name: secret-volume
            mountPath: /etc/secret-volume
            readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never
```

シークレットデータと共に環境変数が設定された Pod の YAML

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

シークレットデータと環境変数を設定するビルド設定の YAML

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

2.3.6.4. 入力シークレットおよび設定マップの追加

認証情報およびその他の設定データをソース管理に配置せずにビルドに提供するには、入力シークレットおよび入力設定マップを定義します。

シナリオによっては、ビルド操作で、依存するリソースにアクセスするための認証情報や他の設定データが必要になる場合があります。この情報をソース管理に配置せずに利用可能にするには、入力シークレットおよび入力設定マップを定義します。

手順

既存の **BuildConfig** オブジェクトに入力シークレットおよび/または設定マップを追加するには、以下を行います。

1. **ConfigMap** オブジェクトがない場合はこれを作成します。

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

これにより、**settings-mvn** という名前の新しい設定マップが作成されます。これには、**settings.xml** ファイルのプレーンテキストのコンテンツが含まれます。

ヒント

または、以下の YAML を適用して設定マップを作成できます。

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

2. **Secret** オブジェクトがない場合はこれを作成します。

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

これにより、**secret-mvn** という名前の新規シークレットが作成されます。 これには、 **id_rsa** プライベートキーの base64 でエンコードされたコンテンツが含まれます。

ヒント

または、以下の YAML を適用して入力シークレットを作成できます。

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

3. 設定マップおよびシークレットを既存の **BuildConfig** オブジェクトの **source** セクションに追加します。

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

シークレットおよび設定マップを新規の **BuildConfig** オブジェクトに追加するには、以下のコマンドを実行します。

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

ビルド時に、**settings.xml** および **id_rsa** ファイルはソースコードが配置されているディレクトリーにコピーされます。OpenShift Container Platform S2I ビルダーイメージでは、これはイメージの作業ディレクトリーで、 **Dockerfile** の **WORKDIR** の指示を使用して設定されます。別のディレクトリーを指定するには、 **destinationDir** を定義に追加します。

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

新規の **BuildConfig** オブジェクトの作成時に、宛先のディレクトリーを指定することも可能です。

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

いずれの場合も、**settings.xml** ファイルがビルド環境の **./.m2** ディレクトリーに追加され、**id_rsa** キーは **./.ssh** ディレクトリーに追加されます。

2.3.6.5. Source-to-Image ストラテジー

Source ストラテジーを使用すると、定義された入力シークレットはすべて、適切な **destinationDir** にコピーされます。**destinationDir** を空にすると、シークレットはビルダーイメージの作業ディレクトリーに配置されます。

destinationDir が相対パスの場合に同じルールが使用されます。シークレットは、イメージの作業ディレクトリーに相対的なパスに配置されます。**destinationDir** パスの最終ディレクトリーは、ビルダーイメージにない場合に作成されます。**destinationDir** の先行するすべてのディレクトリーは存在している必要があり、そうでない場合にはエラーが生じます。

注記

入力シークレットは全ユーザーに書き込み権限が割り当てられた状態で追加され (**0666** のパーミッション)、**assemble** スクリプトの実行後には、サイズが 0 になるように切り捨てられます。つまり、シークレットファイルは作成されたイメージ内に存在しますが、セキュリティーの理由で空になります。

入力設定マップは、**assemble** スクリプトの実行後に切り捨てられません。

2.3.6.6. Docker ストラテジー

docker ストラテジーを使用すると、Dockerfile で **ADD** および **COPY** の命令 を使用してコンテナーイメージに定義されたすべての入力シークレットを追加できます。

シークレットの **destinationDir** を指定しない場合は、ファイルは、Dockerfile が配置されているのと同じディレクトリーにコピーされます。相対パスを **destinationDir** として指定する場合は、シークレットは、Dockerfile の場所と相対的なディレクトリーにコピーされます。これにより、ビルド時に使用するコンテキストディレクトリーの一部として、Docker ビルド操作でシークレットファイルが利用できるようになります。

シークレットおよび設定マップデータを参照する Dockerfile の例

```
FROM centos/ruby-22-centos7

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

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

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

-

重要

通常はシークレットがイメージから実行するコンテナーに置かれないように、入力シークレットを最終的なアプリケーションイメージから削除します。ただし、シークレットは追加される階層のイメージ自体に存在します。この削除は、Dockerfile の一部として組み込まれます。

入力シークレットおよび設定マップのコンテンツがビルド出力コンテナーイメージに表示されないようにして、この削除プロセスを完全に回避するには、代わりに Docker ビルドストラテジーで ビルドボリュームを使用 します。

2.3.6.7. カスタムストラテジー

Custom ストラテジーを使用する場合、定義された入力シークレットおよび設定マップはすべて、**/var/run/secrets/openshift.io/build** ディレクトリー内のビルダーコンテナーで入手できます。カスタムのビルドイメージは、これらのシークレットおよび設定マップを適切に使用する必要があります。Custom ストラテジーでは、Custom ストラテジーのオプションで説明されているようにシークレットを定義できます。

既存のストラテジーのシークレットと入力シークレットには違いはありません。ただし、ビルダーイメージはこれらを区別し、ビルドのユースケースに基づいてこれらを異なる方法で使用する場合があります。

入力シークレットは常に **/var/run/secrets/openshift.io/build** ディレクトリーにマウントされます。 そうでない場合には、ビルダーが完全なビルドオブジェクトを含む **$BUILD** 環境変数を解析できます。

重要

レジストリーのプルシークレットが namespace とノードの両方に存在する場合、ビルドがデフォルトで namespace でのプルシークレットの使用に設定されます。

2.3.7. 外部アーティファクト

ソースリポジトリーにバイナリーファイルを保存することは推奨していません。そのため、ビルドプロセス中に追加のファイル (Java **.jar** の依存関係など) をプルするビルドを定義する必要がある場合があります。この方法は、使用するビルドストラテジーにより異なります。

Source ビルドストラテジーの場合は、**assemble** スクリプトに適切なシェルコマンドを設定する必要があります。

.s2i/bin/assemble ファイル

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

.s2i/bin/run ファイル

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

Docker ビルドストラテジーの場合は、Dockerfile を変更して、**RUN** 命令 を指定してシェルコマンドを呼び出す必要があります。

Dockerfile の抜粋

```
FROM jboss/base-jdk:8

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

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

実際には、ファイルの場所の環境変数を使用し、Dockerfile または **assemble** スクリプトを更新するのではなく、**BuildConfig** で定義した環境変数で、ダウンロードする特定のファイルをカスタマイズすることができます。

環境変数の定義には複数の方法があり、いずれかの方法を選択できます。

- **.s2i/environment** ファイルの使用 (ソースビルドストラテジーのみ)
- **BuildConfig** での設定
- **oc start-build --env** を使用した明示的な指定 (手動でトリガーされるビルドのみが対象)

2.3.8. プライベートレジストリーでの docker 認証情報の使用

プライベートコンテナーレジストリーの有効な認証情報を指定して、**.docker/config.json** ファイルでビルドを提供できます。これにより、プライベートコンテナーイメージレジストリーにアウトプットイメージをプッシュしたり、認証を必要とするプライベートコンテナーイメージレジストリーからビルダーイメージをプルすることができます。

同じレジストリー内に、レジストリーパスに固有の認証情報を指定して、複数のリポジトリーに認証情報を指定できます。

注記

OpenShift Container Platform コンテナーイメージレジストリーでは、OpenShift Container Platform が自動的にシークレットを生成するので、この作業は必要ありません。

デフォルトでは、**.docker/config.json** ファイルはホームディレクトリーにあり、以下の形式となっています。

```
auths:
  index.docker.io/v1/: 1
    auth: "YWRfbGzhcGU6R2labnRib21ifTE=" 2
    email: "user@example.com" 3
  docker.io/my-namespace/my-user/my-image: 4
    auth: "GzhYWRGU6R2fbclabnRgbkSp=""
    email: "user@example.com"
  docker.io/my-namespace: 5
    auth: "GzhYWRGU6R2deesfrRgbkSp=""
    email: "user@example.com"
```

1 レジストリーの URL

2 暗号化されたパスワード

3 ログイン用のメールアドレス

4 namespace 内の特定イメージの URL および認証情報

5 レジストリー namespace の URL および認証情報

複数のコンテナーイメージレジストリーを定義するか、同じレジストリーに複数のリポジトリーを定義することができます。または **docker login** コマンドを実行して、このファイルに認証エントリーを追加することも可能です。ファイルが存在しない場合には作成されます。

Kubernetes では **Secret** オブジェクトが提供され、これを使用して設定とパスワードを保存することができます。

前提条件

- **.docker/config.json** ファイルが必要です。

手順

1. ローカルの **.docker/config.json** ファイルからシークレットを作成します。

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

このコマンドにより、**dockerhub** という名前のシークレットの JSON 仕様が生成され、オブジェクトが作成されます。

2. **pushSecret** フィールドを **BuildConfig** の **output** セクションに追加し、作成した **secret** の名前 (上記の例では、**dockerhub**) に設定します。

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

oc set build-secret コマンドを使用して、ビルド設定にプッシュするシークレットを設定します。

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

pushSecret フィールドを指定する代わりに、プッシュシークレットをビルドで使用されるサービスアカウントにリンクできます。デフォルトで、ビルドは **builder** サービスアカウントを使用します。シークレットにビルドのアウトプットイメージをホストするリポジトリーに一致する認証情報が含まれる場合、プッシュシークレットはビルドに自動的に追加されます。

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

3. ビルドストラテジー定義に含まれる **pullSecret** を指定して、プライベートコンテナーイメージレジストリーからビルダーコンテナーイメージをプルします。

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

oc set build-secret コマンドを使用して、ビルド設定でプルシークレットを設定します。

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

注記

以下の例では、ソールビルドに **pullSecret** を使用しますが、Docker とカスタムビルドにも該当します。

pullSecret フィールドを指定する代わりに、プルシークレットをビルドで使用されるサービスアカウントにリンクできます。デフォルトで、ビルドは **builder** サービスアカウントを使用します。シークレットにビルドのインプットイメージをホストするリポジトリーに一致する認証情報が含まれる場合、プルシークレットはビルドに自動的に追加されます。**pullSecret** フィールドを指定する代わりに、プルシークレットをビルドで使用されるサービスアカウントにリンクするには、以下を実行します。

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

注記

この機能を使用するには、**from** イメージを **BuildConfig** 仕様に指定する必要があります。**oc new-build** または **oc new-app** で生成される Docker ストラテジービルドは、場合によってこれを実行しない場合があります。

2.3.9. ビルド環境

Pod 環境変数と同様に、ビルドの環境変数は Downward API を使用して他のリソースや変数の参照として定義できます。ただし、いくつかは例外があります。

oc set env コマンドで、**BuildConfig** に定義した環境変数を管理することも可能です。

注記

参照はコンテナーの作成前に解決されるため、ビルド環境変数の **valueFrom** を使用したコンテナーリソースの参照はサポートされません。

2.3.9.1. 環境変数としてのビルドフィールドの使用

ビルドオブジェクトの情報は、値を取得するフィールドの **JsonPath** に、**fieldPath** 環境変数のソースを設定することで挿入できます。

注記

Jenkins Pipeline ストラテジーは、環境変数の **valueFrom** 構文をサポートしません。

手順

- 値を取得するフィールドの **JsonPath** に、**fieldPath** 環境変数のソースを設定します。

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

2.3.9.2. 環境変数としてのシークレットの使用

valueFrom 構文を使用して、シークレットからのキーの値を環境変数として利用できます。

重要

この方法では、シークレットをビルド Pod コンソールの出力でプレーンテキストとして表示します。これを回避するには、代わりに入力シークレットおよび設定マップを使用します。

手順

- シークレットを環境変数として使用するには、**valueFrom** 構文を設定します。

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

関連情報

- 入力シークレットおよび設定マップ

2.3.10. サービス提供証明書のシークレット

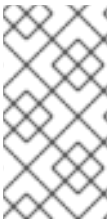
サービスが提供する証明書のシークレットは、追加設定なしの証明書を必要とする複雑なミドルウェアアプリケーションをサポートするように設計されています。これにはノードおよびマスターの管理者ツールで生成されるサーバー証明書と同じ設定が含まれます。

手順

サービスとの通信のセキュリティーを保護するには、クラスターが署名された提供証明書/キーペアを namespace のシークレットに生成できるようにします。

- 値をシークレットに使用する名前に設定し、**service.beta.openshift.io/serving-cert-secret-name** アノテーションをサービスに設定します。
  次に、**PodSpec** はそのシークレットをマウントできます。これが利用可能な場合、Pod が実行されます。この証明書は内部サービス DNS 名、**<service.name>.<service.namespace>.svc** に適しています。

  証明書およびキーは PEM 形式であり、それぞれ **tls.crt** および **tls.key** に保存されます。証明書/キーのペアは有効期限に近づくと自動的に置換されます。シークレットの **service.beta.openshift.io/expiry** アノテーションで RFC3339 形式の有効期限の日付を確認します。



注記

ほとんどの場合、サービス DNS 名 **<service.name>.<service.namespace>.svc** は外部にルーティング可能ではありません。**<service.name>.<service.namespace>.svc** の主な使用方法として、クラスターまたはサービス間の通信用として、 re-encrypt ルートで使用されます。

他の Pod は Pod に自動的にマウントされる **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** ファイルの認証局 (CA) バンドルを使用して、クラスターで作成される証明書 (内部 DNS 名の場合にのみ署名される) を信頼できます。

この機能の署名アルゴリズムは **x509.SHA256WithRSA** です。ローテーションを手動で実行するには、生成されたシークレットを削除します。新規の証明書が作成されます。

2.3.11. シークレットの制限

シークレットを使用するには、Pod がシークレットを参照できる必要があります。シークレットは、以下の 3 つの方法で Pod で使用されます。

- コンテナーの環境変数を事前に設定するために使用される。
- 1 つ以上のコンテナーにマウントされるボリュームのファイルとして使用される。
- Pod のイメージをプルする際に kubelet によって使用される。

ボリュームタイプのシークレットは、ボリュームメカニズムを使用してデータをファイルとしてコンテナーに書き込みます。**imagePullSecrets** は、シークレットを namespace のすべての Pod に自動的に挿入するためにサービスアカウントを使用します。

テンプレートにシークレット定義が含まれる場合、テンプレートで指定のシークレットを使用できるようにするには、シークレットのボリュームソースを検証し、指定されるオブジェクト参照が **Secret** タイプのオブジェクトを実際に参照していることを確認できる必要があります。そのため、シークレットはこれに依存する Pod の作成前に作成されている必要があります。最も効果的な方法として、サービスアカウントを使用してシークレットを自動的に挿入することができます。

シークレット API オブジェクトは namespace にあります。それらは同じ namespace の Pod によってのみ参照されます。

個々のシークレットは 1MB のサイズに制限されます。これにより、apiserver および kubelet メモリーを使い切るような大規模なシークレットの作成を防ぐことができます。ただし、小規模なシークレットであってもそれらを数多く作成するとメモリーの消費につながります。

2.4. ビルド出力の管理

ビルド出力の概要およびビルド出力の管理方法についての説明については、以下のセクションを使用します。

2.4.1. ビルド出力

docker または Source-to-Image (S2I) ストラテジーを使用するビルドにより、新しいコンテナーイメージが作成されます。このイメージは、**Build** 仕様の **output** セクションで指定されているコンテナーイメージのレジストリーにプッシュされます。

出力の種類が **ImageStreamTag** の場合は、イメージが統合された OpenShift イメージレジストリーにプッシュされ、指定のイメージストリームにタグ付けされます。出力が **DockerImage** タイプの場合は、出力参照の名前が docker のプッシュ仕様として使用されます。この仕様にレジストリーが含まれる場合もありますが、レジストリーが指定されていない場合は、DockerHub にデフォルト設定されます。ビルド仕様の出力セクションが空の場合には、ビルドの最後にイメージはプッシュされません。

ImageStreamTag への出力

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

docker のプッシュ仕様への出力

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

2.4.2. アウトプットイメージの環境変数

docker および Source-to-Image (S2I) ストラテジービルドは、以下の環境変数をアウトプットイメージに設定します。

変数	説明
OPENSHIFT_BUILD_NAME	ビルドの名前
OPENSHIFT_BUILD_NAMESPACE	ビルドの namespace
OPENSHIFT_BUILD_SOURCE	ビルドのソース URL
OPENSHIFT_BUILD_REFERENCE	ビルドで使用する Git 参照
OPENSHIFT_BUILD_COMMIT	ビルドで使用するソースコミット

また、S2I] または docker ストラテジーオプションなどで設定されたユーザー定義の環境変数も、アウトプットイメージの環境変数リストの一部になります。

2.4.3. アウトプットイメージのラベル

docker および Source-to-Image (S2I) ビルドは、以下のラベルをアウトプットイメージに設定します。

ラベル	説明
io.openshift.build.commit.author	ビルドで使用するソースコミットの作成者
io.openshift.build.commit.date	ビルドで使用するソースコミットの日付
io.openshift.build.commit.id	ビルドで使用するソースコミットのハッシュ
io.openshift.build.commit.message	ビルドで使用するソースコミットのメッセージ
io.openshift.build.commit.ref	ソースに指定するブランチまたは参照
io.openshift.build.source-location	ビルドのソース URL

BuildConfig.spec.output.imageLabels フィールドを使用して、カスタムラベルのリストを指定することも可能です。 このラベルは、ビルド設定の各イメージビルドに適用されます。

ビルドイメージに適用されるカスタムラベル

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

2.5. ビルドストラテジーの使用

以下のセクションでは、主なサポートされているビルドストラテジー、およびそれらの使用方法を定義します。

2.5.1. Docker ビルド

OpenShift Container Platform は Buildah を使用して Dockerfile からコンテナーイメージをビルドします。Dockerfile を使用したコンテナーイメージのビルドについての詳細は、Dockerfile リファレンスドキュメント を参照してください。

ヒント

buildArgs 配列を使用して Docker ビルド引数を設定する場合は、Dockerfile リファレンスドキュメントの ARG および FROM の対話方法 について参照してください。

2.5.1.1. Dockerfile FROM イメージの置き換え

Dockerfile の **FROM** 命令は、**BuildConfig** オブジェクトの **from** に置き換えられます。Dockerfile がマルチステージビルドを使用する場合、最後の **FROM** 命令のイメージを置き換えます。

手順

Dockerfile の **FROM** 命令は、**BuildConfig** の **from** に置き換えられます。

```
strategy:
  dockerStrategy:
    from:
      kind: "ImageStreamTag"
      name: "debian:latest"
```

2.5.1.2. Dockerfile パスの使用

デフォルトで、docker ビルドは、**BuildConfig.spec.source.contextDir** フィールドで指定されたコンテキストのルートに配置されている Dockerfile を使用します。

dockerfilePath フィールドでは、ビルドが異なるパスを使用して Dockerfile ファイルの場所 (**BuildConfig.spec.source.contextDir** フィールドへの相対パス) を特定できます。デフォルトの Dockerfile (例: **MyDockerfile**) とは異なるファイル名や、サブディレクトリーにある Dockerfile へのパス (例: **dockerfiles/app1/Dockerfile**) を設定できます。

手順

ビルドが Dockerfile を見つけるために異なるパスを使用できるように **dockerfilePath** フィールドを使用するには、以下を設定します。

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

2.5.1.3. docker 環境変数の使用

環境変数を docker ビルドプロセスおよび結果として生成されるイメージで利用可能にするには、環境変数をビルド設定の **dockerStrategy** 定義に追加できます。

ここに定義した環境変数は、Dockerfile 内で後に参照できるよう単一の **ENV** Dockerfile 命令として **FROM** 命令の直後に挿入されます。

手順

変数はビルド時に定義され、アウトプットイメージに残るため、そのイメージを実行するコンテナーにも存在します。

たとえば、ビルドやランタイム時にカスタムの HTTP プロキシーを定義するには以下を設定します。

```
dockerStrategy:
```

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

oc set env コマンドで、ビルド設定に定義した環境変数を管理することも可能です。

2.5.1.4. docker ビルド引数の追加

buildArgs 配列を使用して docker ビルド引数 を設定できます。ビルド引数は、ビルドの開始時に docker に渡されます。

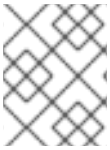
ヒント

Dockerfile リファレンスドキュメントの Understand how ARG and FROM interact を参照してください。

手順

docker ビルドの引数を設定するには、以下のように **buildArgs** 配列にエントリーを追加します。これは、**BuildConfig** オブジェクトの **dockerStrategy** 定義の中にあります。以下に例を示します。

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



注記

name および **value** フィールドのみがサポートされます。**valueFrom** フィールドの設定は無視されます。

2.5.1.5. Docker ビルドによる層の非表示

Docker ビルドは通常、Dockerfile のそれぞれの命令を表す層を作成します。**imageOptimizationPolicy** を **SkipLayers** に設定することにより、すべての命令がベースイメージ上部の単一層にマージされます。

手順

- **imageOptimizationPolicy** を **SkipLayers** に設定します。

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

2.5.1.6. ビルドボリュームの使用

ビルドボリュームをマウントして、実行中のビルドに、アウトプットコンテナーイメージで永続化しない情報にアクセスできます。

ビルドボリュームは、ビルド時にビルド環境や設定が必要なリポジトリーの認証情報など、機密情報のみを提供します。ビルドボリュームは、データが出力コンテナーイメージに保持される ビルド入力 とは異なります。

実行中のビルドがデータを読み取るビルドボリュームのマウントポイントは機能的に pod volume mounts に似ています。

前提条件

- 入力シークレット、設定マップ、またはその両方を BuildConfig オブジェクトに追加している。

手順

- **BuildConfig** オブジェクトの **dockerStrategy** 定義で、ビルドボリュームを **volumes** 配列に追加します。以下に例を示します。

```
spec:
  dockerStrategy:
    volumes:
      - name: secret-mvn 1
        mounts:
        - destinationPath: /opt/app-root/src/.ssh 2
        source:
          type: Secret 3
          secret:
            secretName: my-secret 4
      - name: settings-mvn 5
        mounts:
        - destinationPath: /opt/app-root/src/.m2 6
        source:
          type: ConfigMap 7
          configMap:
            name: my-config 8
      - name: my-csi-volume 9
        mounts:
        - destinationPath: /opt/app-root/src/some_path 10
        source:
          type: CSI 11
          csi:
            driver: csi.sharedresource.openshift.io 12
            readOnly: true 13
            volumeAttributes: 14
              attribute: value
```

1 5 9 必須。一意な名前

2 6 10 必須。マウントポイントの絶対パス。**..** または **:** を含めないでください。こうすることで、ビルダーが生成した宛先パスと競合しなくなります。**/opt/app-root/src** は、多くの Red Hat S2I 対応イメージのデフォルトのホームディレクトリーです。

3 7 11 必須。ソースのタイプは、**ConfigMap**、**Secret**、または **CSI**。

4 8 必須。ソースの名前。

12 必須。一時 CSI ボリュームを提供するドライバー。

13 オプション: true の場合、ドライバーに読み取り専用ボリュームを提供するように指示します。

14 オプション: 一時 CSI ボリュームのボリューム属性。サポートされる属性キーおよび値については、CSI ドライバーのドキュメントを参照してください。

注記

共有リソース CSI ドライバーは、テクノロジープレビュー機能としてサポートされています。

2.5.2. Source-to-Image ビルド

Source-to-Image (S2I) は再現可能なコンテナーイメージをビルドするためのツールです。これはアプリケーションソースをコンテナーイメージに挿入し、新規イメージをアセンブルして実行可能なイメージを生成します。新規イメージはベースイメージ、ビルダーおよびビルドされたソースを組み込み、**buildah run** コマンドで使用することができます。S2I は増分ビルドをサポートします。これは以前にダウンロードされた依存関係や、以前にビルドされたアーティファクトなどを再利用します。

2.5.2.1. Source-to-Image (S2I) 増分ビルドの実行

Source-to-Image (S2I) は増分ビルドを実行できます。 つまり、以前にビルドされたイメージからアーティファクトが再利用されます。

手順

- 増分ビルドを作成するには、ストラテジー定義に以下の変更を加えてこれを作成します。

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "incremental-image:latest" 1
    incremental: true 2
```

1 増分ビルドをサポートするイメージを指定します。この動作がサポートされているか判断するには、ビルダーイメージのドキュメントを参照してください。

2 このフラグでは、増分ビルドを試行するかどうかを制御します。ビルダーイメージで増分ビルドがサポートされていない場合は、ビルドは成功しますが、**save-artifacts** スクリプトがないため、増分ビルドに失敗したというログメッセージが表示されます。

関連情報

- 増分ビルドをサポートするビルダーイメージを作成する方法の詳細については、S2I 要件について参照してください。

2.5.2.2. Source-to-Image (S2I) ビルダーイメージスクリプトの上書き

ビルダーイメージによって提供される **assemble**、**run**、および **save-artifacts** Source-to-Image (S2I) スクリプトを上書きできます。

手順

ビルダーイメージによって提供される **assemble**、**run**、および **save-artifacts** S2I スクリプトを上書きするには、以下のいずれかを実行します。

- アプリケーションのソースリポジトリーの **.s2i/bin** ディレクトリーに **assemble**、**run**、 または **save-artifacts** スクリプトを指定します。
- ストラテジー定義の一部として、スクリプトを含むディレクトリーの URL を指定します。以下に例を示します。

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

1 このパスに、**run**、**assemble**、および **save-artifacts** が追加されます。一部または全スクリプトがある場合、そのスクリプトが、イメージに指定された同じ名前のスクリプトの代わりに使用されます。

注記

scripts URL にあるファイルは、ソースリポジトリーの **.s2i/bin** にあるファイルよりも優先されます。

2.5.2.3. Source-to-Image 環境変数

ソースビルドのプロセスと生成されるイメージで環境変数を利用できるようにする方法として、2 つの方法があります。2 種類 (環境ファイルおよび BuildConfig 環境の値の使用) あります。指定される変数は、ビルドプロセスでアウトプットイメージに表示されます。

2.5.2.3.1. Source-to-Image 環境ファイルの使用

ソースビルドでは、ソースリポジトリーの **.s2i/environment** ファイルに指定することで、アプリケーション内に環境の値 (1 行に 1 つ) を設定できます。このファイルに指定される環境変数は、ビルドプロセス時にアウトプットイメージに表示されます。

ソースリポジトリーに **.s2i/environment** ファイルを渡すと、Source-to-Image (S2I) はビルド時にこのファイルを読み取ります。これにより **assemble** スクリプトがこれらの変数を使用できるので、ビルドの動作をカスタマイズできます。

手順

たとえば、ビルド中の Rails アプリケーションのアセットコンパイルを無効にするには、以下を実行します。

- **DISABLE_ASSET_COMPILATION=true** を **.s2i/environment** ファイルに追加します。

ビルド以外に、指定の環境変数も実行中のアプリケーション自体で利用できます。たとえば、Rails アプリケーションが **production** ではなく **development** モードで起動できるようにするには、以下を実行します。

- **RAILS_ENV=development** を **.s2i/environment** ファイルに追加します。

サポートされる環境変数の完全なリストについては、各イメージのイメージの使用についてのセクションを参照してください。

2.5.2.3.2. Source-to-Image ビルド設定環境の使用

環境変数をビルド設定の **sourceStrategy** 定義に追加できます。ここに定義されている環境変数は、**assemble** スクリプトの実行時に表示され、アウトプットイメージで定義されるので、**run** スクリプトやアプリケーションコードでも利用できるようになります。

手順

- たとえば、Rails アプリケーションのアセットコンパイルを無効にするには、以下を実行します。

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

関連情報

- ビルド環境のセクションでは、より詳細な説明を提供します。
- **oc set env** コマンドで、ビルド設定に定義した環境変数を管理することも可能です。

2.5.2.4. Source-to-Image ソースファイルを無視する

Source-to-Image (S2I) は **.s2iignore** ファイルをサポートします。これには、無視する必要のあるファイルパターンのリストが含まれます。このファイルには、無視すべきファイルパターンのリストが含まれます。 **.s2iignore** ファイルにあるパターンと一致する、さまざまな入力ソースで提供されるビルドの作業ディレクトリーにあるファイルは **assemble** スクリプトでは利用できません。

2.5.2.5. Source-to-Image によるソースコードからのイメージの作成

Source-to-Image (S2I) は、アプリケーションのソースコードを入力として取り、アセンブルされたアプリケーションを出力として実行する新規イメージを生成するイメージを簡単に作成できるようにするフレームワークです。

再生成可能なコンテナーイメージのビルドに S2I を使用する主な利点として、開発者の使い勝手の良さが挙げられます。ビルダーイメージの作成者は、イメージが最適な S2I パフォーマンスを実現できるように、ビルドプロセスと S2I スクリプトの基本的なコンセプト 2 点を理解する必要があります。

2.5.2.5.1. Source-to-Image ビルドプロセスについて

ビルドプロセスは、以下の 3 つの要素で設定されており、これら 3 つを組み合わせて最終的なコンテナーイメージが作成されます。

- ソース

- Source-to-Image (S2I) スクリプト
- ビルダーイメージ

S2I は、最初の **FROM** 命令として、ビルダーイメージで Dockerfile を生成します。S2I によって生成される Dockerfile は Buildah に渡されます。

2.5.2.5.2. Source-to-Image スクリプトの作成方法

Source-to-Image (S2I) スクリプトは、ビルダーイメージ内でスクリプトを実行できる限り、どのプログラム言語でも記述できます。S2I は **assemble**/**run**/**save-artifacts** スクリプトを提供する複数のオプションをサポートします。ビルドごとに、これらの場所はすべて、以下の順番にチェックされます。

1. ビルド設定に指定されるスクリプト
2. アプリケーションソースの **.s2i**/**bin** ディレクトリーにあるスクリプト
3. **io.openshift.s2i.scripts-url** ラベルを含むデフォルトの URL にあるスクリプト

イメージで指定した **io.openshift.s2i.scripts-url** ラベルも、ビルド設定で指定したスクリプトも、以下の形式のいずれかを使用します。

- **image:///path_to_scripts_dir**: S2I スクリプトが配置されているディレクトリーへのイメージ内の絶対パス。
- **file:///path_to_scripts_dir**: S2I スクリプトが配置されているディレクトリーへのホスト上の相対パスまたは絶対パス。
- **http(s)://path_to_scripts_dir**: S2I スクリプトが配置されているディレクトリーの URL。

表2.1 S2I スクリプト

スクリプト	説明
assemble	**assemble** スクリプトは、ソースからアプリケーションアーティファクトをビルドし、イメージ内の適切なディレクトリーに配置します。このスクリプトが必要です。このスクリプトのワークフローは以下のとおりです。 1. オプション: ビルドのアーティファクトを復元します。増分ビルドをサポートする必要がある場合、**save-artifacts** も定義するようにしてください (オプション)。 2. 任意の場所に、アプリケーションソースを配置します。 3. アプリケーションのアーティファクトをビルドします。 4. 実行に適した場所に、アーティファクトをインストールします。
run	**run** スクリプトはアプリケーションを実行します。このスクリプトが必要です。

スクリプト	説明
save-artifacts	**save-artifacts** スクリプトは、次に続くビルドプロセスを加速できるようにすべての依存関係を収集します。このスクリプトはオプションです。以下に例を示します。 ● Ruby の場合は、Bundler でインストールされる **gems** ● Java の場合は、**.m2** のコンテンツ これらの依存関係は **tar** ファイルに集められ、標準出力としてストリーミングされます。
usage	**usage** スクリプトでは、ユーザーに、イメージの正しい使用方法を通知します。このスクリプトはオプションです。
test/run	**test/run** スクリプトでは、イメージが正しく機能しているかどうかを確認するためのプロセスを作成できます。このスクリプトはオプションです。このプロセスの推奨フローは以下のとおりです。 1. イメージをビルドします。 2. イメージを実行して **usage** スクリプトを検証します。 3. **s2i build** を実行して **assemble** スクリプトを検証します。 4. オプション: 再度 **s2i build** を実行して、**save-artifacts** と **assemble** スクリプトの保存、復元アーティファクト機能を検証します。 5. イメージを実行して、テストアプリケーションが機能していることを確認します。  **注記** **test/run** スクリプトでビルドしたテストアプリケーションを配置するための推奨される場所は、イメージリポジトリーの **test/test-app** ディレクトリーです。

S2I スクリプトの例

以下の S2I スクリプトの例は Bash で記述されています。それぞれの例では、**tar** の内容は **/tmp/s2i** ディレクトリーにデプロイメントされることが前提とされています。

assemble スクリプト:

```
#!/bin/bash

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

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

```
# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run スクリプト:

```
#!/bin/bash

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

save-artifacts スクリプト:

```
#!/bin/bash

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

usage スクリプト:

```
#!/bin/bash

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

関連情報

- S2I イメージ作成のチュートリアル

2.5.2.6. ビルドボリュームの使用

ビルドボリュームをマウントして、実行中のビルドに、アウトプットコンテナーイメージで永続化しない情報にアクセスできます。

ビルドボリュームは、ビルド時にビルド環境や設定が必要なリポジトリーの認証情報など、機密情報のみを提供します。ビルドボリュームは、データが出力コンテナーイメージに保持される ビルド入力 とは異なります。

実行中のビルドがデータを読み取るビルドボリュームのマウントポイントは機能的に pod volume mounts に似ています。

前提条件

前提条件

- 入力シークレット、設定マップ、またはその両方を BuildConfig オブジェクトに追加している。

手順

- **BuildConfig** オブジェクトの **sourceStrategy** 定義で、ビルドボリュームを **volumes** 配列に追加します。以下に例を示します。

```
spec:
  sourceStrategy:
    volumes:
      - name: secret-mvn (1)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh (2)
        source:
          type: Secret (3)
          secret:
            secretName: my-secret (4)
      - name: settings-mvn (5)
        mounts:
        - destinationPath: /opt/app-root/src/.m2 (6)
        source:
          type: ConfigMap (7)
          configMap:
            name: my-config (8)
      - name: my-csi-volume (9)
        mounts:
        - destinationPath: /opt/app-root/src/some_path (10)
        source:
          type: CSI (11)
          csi:
            driver: csi.sharedresource.openshift.io (12)
            readOnly: true (13)
            volumeAttributes: (14)
              attribute: value
```

(1) (5) (9) 必須。一意な名前

(2) (6) (10) 必須。マウントポイントの絶対パス。**..** または **:** を含めないでください。こうすることで、ビルダーが生成した宛先パスと競合しなくなります。**/opt/app-root/src** は、多くの Red Hat S2I 対応イメージのデフォルトのホームディレクトリーです。

(3) (7) (11) 必須。ソースのタイプは、**ConfigMap**、**Secret**、または **CSI**。

(4) (8) 必須。ソースの名前。

(12) 必須。一時 CSI ボリュームを提供するドライバー。

(13) オプション: true の場合、ドライバーに読み取り専用ボリュームを提供するように指示します。

(14) オプション: 一時 CSI ボリュームのボリューム属性。サポートされる属性キーおよび値については、CSI ドライバーのドキュメントを参照してください。

注記

共有リソース CSI ドライバーは、テクノロジープレビュー機能としてサポートされています。

2.5.3. カスタムビルド

カスタムビルドストラテジーにより、開発者はビルドプロセス全体を対象とする特定のビルダーイメージを定義できます。独自のビルダーイメージを使用することにより、ビルドプロセスをカスタマイズできます。

カスタムビルダーイメージは、RPM またはベースイメージの構築など、ビルドプロセスのロジックに組み込まれるプレーンなコンテナーイメージです。

カスタムビルドは高いレベルの権限で実行されるため、デフォルトではユーザーが利用することはできません。クラスター管理者のパーミッションを持つ信頼できるユーザーのみにカスタムビルドを実行するためのアクセスが付与される必要があります。

2.5.3.1. カスタムビルドの FROM イメージの使用

customStrategy.from セクションを使用して、カスタムビルドに使用するイメージを指定できます。

手順

- **customStrategy.from** セクションを設定するには、以下を実行します。

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

2.5.3.2. カスタムビルドでのシークレットの使用

すべてのビルドタイプに追加できるソースおよびイメージのシークレットのほかに、カスタムストラテジーを使用することにより、シークレットの任意のリストをビルダー Pod に追加できます。

手順

- 各シークレットを特定の場所にマウントするには、**strategy** YAML ファイルの **secretSource** および **mountPath** フィールドを編集します。

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

1 **secretSource** は、ビルドと同じ namespace にあるシークレットへの参照です。

2 **mountPath** は、シークレットがマウントされる必要のあるカスタムビルダー内のパスです。

2.5.3.3. カスタムビルドの環境変数の使用

環境変数をカスタムビルドプロセスで利用可能にするには、環境変数をビルド設定の **customStrategy** 定義に追加できます。

ここに定義された環境変数は、カスタムビルドを実行する Pod に渡されます。

手順

1. ビルド時に使用されるカスタムの HTTP プロキシーを定義します。

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

2. ビルド設定で定義された環境変数を管理するには、以下のコマンドを入力します。

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

2.5.3.4. カスタムビルダーイメージの使用

OpenShift Container Platform のカスタムビルドストラテジーにより、ビルドプロセス全体を対象とする特定のビルダーイメージを定義できます。パッケージ、JAR、WAR、インストール可能な ZIP、ベースイメージなどの個別のアーティファクトを生成するためにビルドが必要な場合は、カスタムビルドストラテジーを使用してカスタムビルダーイメージを使用します。

カスタムビルダーイメージは、RPM またはベースのコンテナーイメージの構築など、ビルドプロセスのロジックに組み込まれるプレーンなコンテナーイメージです。

さらに、カスタムビルダーは、単体または統合テストを実行する CI/CD フローなどの拡張ビルドプロセスを実装できます。

2.5.3.4.1. カスタムビルダーイメージ

呼び出し時に、カスタムのビルダーイメージは、ビルドの続行に必要な情報が含まれる以下の環境変数を受け取ります。

表2.2 カスタムビルダーの環境変数

変数名	説明
BUILD	**Build** オブジェクト定義のシリアル化された JSON すべて。シリアル化した中で固有の API バージョンを使用する必要がある場合は、ビルド設定のカスタムストラテジーの仕様で、**buildAPIVersion** パラメーターを設定できます。
SOURCE_REPOSITORY	ビルドするソースが含まれる Git リポジトリーの URL

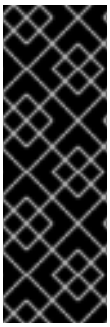
変数名	説明
SOURCE_URI	**SOURCE_REPOSITORY** と同じ値を仕様します。どちらでも使用できます。
SOURCE_CONTEXT_DIR	ビルド時に使用する Git リポジトリーのサブディレクトリーを指定します。定義された場合にのみ表示されます。
SOURCE_REF	ビルドする Git 参照
ORIGIN_VERSION	このビルドオブジェクトを作成した OpenShift Container Platform のマスターのバージョン
OUTPUT_REGISTRY	イメージをプッシュするコンテナーイメージレジストリー
OUTPUT_IMAGE	ビルドするイメージのコンテナーイメージタグ名
PUSH_DOCKERCFG_PATH	**podman push** 操作を実行するためのコンテナーレジストリー認証情報へのパス

2.5.3.4.2. カスタムビルダーのワークフロー

カスタムビルダーイメージの作成者は、ビルドプロセスを柔軟に定義できますが、ビルダーイメージは、OpenShift Container Platform 内でビルドを実行するために必要な以下の手順に従う必要があります。

1. **Build** オブジェクト定義に、ビルドの入力パラメーターの必要情報をすべて含める。
2. ビルドプロセスを実行する。
3. ビルドでイメージが生成される場合には、ビルドの出力場所が定義されていれば、その場所にプッシュする。他の出力場所には環境変数を使用して渡すことができます。

2.5.4. パイプラインビルド



重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

開発者は、パイプラインビルドストラテジーを利用して Jenkins パイプラインプラグインで使用できるように Jenkins パイプラインを定義することができます。このビルドについては、他のビルドタイプの場合と同様に OpenShift Container Platform での起動、モニタリング、管理が可能です。

パイプラインワークフローは、ビルド設定に直接組み込むか、Git リポジトリーに配置してビルド設定で参照して **jenkinsfile** で定義します。

2.5.4.1. OpenShift Container Platform Pipeline について

重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

Pipeline により、OpenShift Container Platform でのアプリケーションのビルド、デプロイ、およびプロモートに対する制御が可能になります。Jenkins Pipeline ビルドストラテジー、**jenkinsfiles**、および OpenShift Container Platform のドメイン固有言語 (DSL) (Jenkins クライアントプラグインで提供される) の組み合わせを使用することにより、すべてのシナリオにおける高度なビルド、テスト、デプロイおよびプロモート用のパイプラインを作成できます。

OpenShift Container Platform Jenkins 同期プラグイン

OpenShift Container Platform Jenkins 同期プラグインは、ビルド設定およびビルドオブジェクトを Jenkins ジョブおよびビルドと同期し、以下を提供します。

- Jenkins での動的なジョブおよび実行の作成。
- イメージストリーム、イメージストリームタグまたは設定マップからのエージェント Pod テンプレートの動的作成。
- 環境変数の挿入。
- OpenShift Container Platform Web コンソールでのパイプラインの可視化。
- Jenkins Git プラグインとの統合。これにより、OpenShift Container Platform ビルドからの Jenkins Git プラグインにコミット情報が渡されます。
- シークレットを Jenkins 認証情報エントリーに同期。

OpenShift Container Platform Jenkins クライアントプラグイン

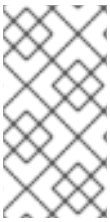
OpenShift Container Platform Jenkins Client プラグインは、OpenShift Container Platform API Server との高度な対話を実現するために、読み取り可能かつ簡潔で、包括的で Fluent (流れるような) スタイルの Jenkins Pipeline 構文を提供することを目的とした Jenkins プラグインです。このプラグインは、スクリプトを実行するノードで使用できる必要がある OpenShift Container Platform コマンドラインツール (**oc**) を使用します。

OpenShift Jenkins クライアントプラグインは Jenkins マスターにインストールされ、OpenShift Container Platform DSL がアプリケーションの **jenkinsfile** 内で利用可能である必要があります。このプラグインは、OpenShift Container Platform Jenkins イメージの使用時にデフォルトでインストールされ、有効にされます。

プロジェクト内で OpenShift Container Platform Pipeline を使用するには、Jenkins Pipeline ビルドストラテジーを使用する必要があります。このストラテジーはソースリポジトリーのルートで **jenkinsfile** を使用するようにデフォルト設定されますが、以下の設定オプションも提供します。

- ビルド設定内のインラインの **jenkinsfile** フィールド。

- ソース **contextDir** との関連で使用する **jenkinsfile** の場所を参照するビルド設定内の **jenkinsfilePath** フィールド。



注記

オプションの **jenkinsfilePath** フィールドは、ソース **contextDir** との関連で使用するファイルの名前を指定します。**contextDir** が省略される場合、デフォルトはリポジトリーのルートに設定されます。**jenkinsfilePath** が省略される場合、デフォルトは **jenkinsfile** に設定されます。

2.5.4.2. パイプラインビルド用の Jenkins ファイルの提供

重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

jenkinsfile は標準的な groovy 言語構文を使用して、アプリケーションの設定、ビルド、およびデプロイメントに対する詳細な制御を可能にします。

jenkinsfile は以下のいずれかの方法で指定できます。

- ソースコードリポジトリー内にあるファイルの使用。
- **jenkinsfile** フィールドを使用してビルド設定の一部として組み込む。

最初のオプションを使用する場合、**jenkinsfile** を以下の場所のいずれかでアプリケーションソースコードリポジトリーに組み込む必要があります。

- リポジトリーのルートにある **jenkinsfile** という名前のファイル。
- リポジトリーのソース **contextDir** のルートにある **jenkinsfile** という名前のファイル。
- ソース **contextDir** に関連して BuildConfig の **JenkinsPipelineStrategy** セクションの **jenkinsfilePath** フィールドで指定される名前のファイル (指定される場合)。 指定されない場合は、リポジトリーのルートにデフォルト設定されます。

jenkinsfile は Jenkins エージェント Pod で実行されます。 ここでは OpenShift Container Platform DSL を使用する場合に OpenShift Container Platform クライアントのバイナリーを利用可能にしておく必要があります。

手順

Jenkins ファイルを指定するには、以下のいずれかを実行できます。

- ビルド設定に Jenkins ファイルを埋め込む
- Jenkins ファイルを含む Git リポジトリーへの参照をビルド設定に追加する

埋め込み定義

■

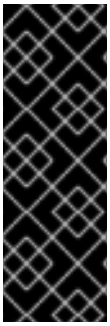
```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('agent') {
          stage 'build'
          openshiftBuild(buildConfig: 'ruby-sample-build', showBuildLogs: 'true')
          stage 'deploy'
          openshiftDeploy(deploymentConfig: 'frontend')
        }
```

Git リポジトリーへの参照

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

1 オプションの **jenkinsfilePath** フィールドは、ソース **contextDir** との関連で使用するファイルの名前を指定します。**contextDir** が省略される場合、デフォルトはリポジトリーのルートに設定されます。**jenkinsfilePath** が省略される場合、デフォルトは **jenkinsfile** に設定されます。

2.5.4.3. Pipeline ビルドの環境変数の使用



重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

環境変数を Pipeline ビルドプロセスで利用可能にするには、環境変数をビルド設定の **jenkinsPipelineStrategy** 定義に追加できます。

定義した後に、環境変数はビルド設定に関連する Jenkins ジョブのパラメーターとして設定されます。

手順

- ビルド時に使用される環境変数を定義するには、YAML ファイルを編集します。

```
jenkinsPipelineStrategy:
...
  env:
    - name: "FOO"
      value: "BAR"
```

oc set env コマンドで、ビルド設定に定義した環境変数を管理することも可能です。

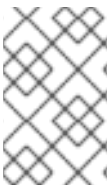
2.5.4.3.1. BuildConfig 環境変数と Jenkins ジョブパラメーター間のマッピング

Pipeline ストラテジーのビルド設定への変更に従い、Jenkins ジョブが作成/更新されると、ビルド設定の環境変数は Jenkins ジョブパラメーターの定義にマッピングされます。 Jenkins ジョブパラメーター定義のデフォルト値は、関連する環境変数の現在の値になります。

Jenkins ジョブの初回作成後に、パラメーターを Jenkins コンソールからジョブに追加できます。パラメーター名は、ビルド設定の環境変数名とは異なります。上記の Jenkins ジョブ用にビルドを開始すると、これらのパラメーターが使用されます。

Jenkins ジョブのビルドを開始する方法により、パラメーターの設定方法が決まります。

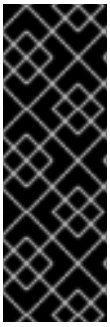
- **oc start-build** で開始された場合には、ビルド設定の環境変数が対応するジョブインスタンスに設定するパラメーターになります。Jenkins コンソールからパラメーターのデフォルト値に変更を加えても無視されます。ビルド設定値が優先されます。
- **oc start-build -e** で開始する場合、**-e** オプションで指定される環境変数の値が優先されます。
  - ビルド設定にリスト表示されていない環境変数を指定する場合、それらは Jenkins ジョブパラメーター定義として追加されます。
  - Jenkins コンソールから環境変数に対応するパラメーターに加える変更は無視されます。ビルド設定および **oc start-build -e** で指定する内容が優先されます。
- Jenkins コンソールで Jenkins ジョブを開始した場合には、ジョブのビルドを開始する操作の一環として、Jenkins コンソールを使用してパラメーターの設定を制御できます。



注記

ジョブパラメーターに関連付けられる可能性のあるすべての環境変数を、ビルド設定に指定することが推奨されます。これにより、ディスク I/O が減り、Jenkins 処理時のパフォーマンスが向上します。

2.5.4.4. Pipeline ビルドのチュートリアル



重要

パイプラインビルドストラテジーは OpenShift Container Platform 4 では非推奨になりました。同等の機能および改善機能は、Tekton をベースとする OpenShift Container Platform Pipeline にあります。

OpenShift Container Platform の Jenkins イメージは完全にサポートされており、ユーザーは Jenkins ユーザーのドキュメントに従ってジョブで **jenkinsfile** を定義するか、これをソースコントロール管理システムに保存します。

以下の例では、**nodejs-mongodb.json** テンプレートを使用して **Node.js/MongoDB** アプリケーションをビルドし、デプロイし、検証する OpenShift Container Platform Pipeline を作成する方法を紹介します。

手順

1. Jenkins マスターを作成するには、以下を実行します。

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

oc new-project <project_name> で新規プロジェクトを使用するか、作成するプロジェクトを選択します。

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

永続ストレージを使用する場合は、**jenkins-persistent** を代わりに使用します。

2. 以下の内容で **nodejs-sample-pipeline.yaml** という名前のファイルを作成します。

注記

Jenkins Pipeline ストラテジーを使用して **Node.js/MongoDB** のサンプルアプリケーションをビルドし、デプロイし、スケーリングする **BuildConfig** オブジェクトを作成します。

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

3. **jenkinsPipelineStrategy** で **BuildConfig** オブジェクトを作成したら、インラインの **jenkinsfile** を使用して、Pipeline に指示を出します。

注記

この例では、アプリケーションに Git リポジトリーを設定しません。

以下の **jenkinsfile** の内容は、OpenShift Container Platform DSL を使用して Groovy で記述されています。ソースリポジトリーに **jenkinsfile** を追加することが推奨される方法ですが、この例では YAML Literal Style を使用して **BuildConfig** にインラインコンテンツを追加しています。

```
def templatePath = 'https://raw.githubusercontent.com/openshift/nodejs-
ex/master/openshift/templates/nodejs-mongodb.json' 1
def templateName = 'nodejs-mongodb-example' 2
pipeline {
  agent {
    node {
```

```
    label 'nodejs' 3
  }
}
options {
  timeout(time: 20, unit: 'MINUTES') 4
}
stages {
  stage('preamble') {
      steps {
          script {
              openshift.withCluster() {
                  openshift.withProject() {
                      echo "Using project: ${openshift.project()}"
                  }
              }
          }
      }
  }
  stage('cleanup') {
    steps {
      script {
          openshift.withCluster() {
              openshift.withProject() {
                openshift.selector("all", [ template : templateName ]).delete() 5
                if (openshift.selector("secrets", templateName).exists()) { 6
                  openshift.selector("secrets", templateName).delete()
                }
              }
          }
      }
    }
  }
  stage('create') {
    steps {
      script {
          openshift.withCluster() {
              openshift.withProject() {
                openshift.newApp(templatePath) 7
              }
          }
      }
    }
  }
  stage('build') {
    steps {
      script {
          openshift.withCluster() {
              openshift.withProject() {
                def builds = openshift.selector("bc", templateName).related('builds')
                timeout(5) { 8
                  builds.untilEach(1) {
                    return (it.object().status.phase == "Complete")
                  }
                }
              }
```

```
            }
          }
        }
      }
      stage('deploy') {
        steps {
          script {
              openshift.withCluster() {
                  openshift.withProject() {
                    def rm = openshift.selector("dc", templateName).rollout()
                    timeout(5) { 9
                      openshift.selector("dc", templateName).related('pods').untilEach(1) {
                        return (it.object().status.phase == "Running")
                      }
                    }
                  }
              }
          }
        }
      }
      stage('tag') {
        steps {
          script {
              openshift.withCluster() {
                  openshift.withProject() {
                    openshift.tag("${templateName}:latest", "${templateName}-staging:latest") 10
                  }
              }
          }
        }
      }
    }
  }
```

1 使用するテンプレートへのパス

1 2 作成するテンプレート名

3 このビルドを実行する **node.js** のエージェント Pod をスピンアップします。

4 この Pipeline に 20 分間のタイムアウトを設定します。

5 このテンプレートラベルが指定されたものすべてを削除します。

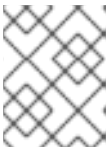
6 このテンプレートラベルが付いたシークレットをすべて削除します。

7 **templatePath** から新規アプリケーションを作成します。

8 ビルドが完了するまで最大 5 分待機します。

9 デプロイメントが完了するまで最大 5 分待機します。

10 すべてが正常に完了した場合は、**$ {templateName}:latest** イメージに **$ {templateName}-staging:latest** のタグを付けます。ステージング環境向けのパイプラインのビルド設定は、変更する **$ {templateName}-staging:latest** イメージがないかを確認し、このイメージをステージング環境にデプロイします。



注記

以前の例は、宣言型のパイプラインスタイルを使用して記述されていますが、以前のスクリプト化されたパイプラインスタイルもサポートされます。

4. OpenShift Container Platform クラスターに Pipeline **BuildConfig** を作成します。

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

   a. 独自のファイルを作成しない場合には、以下を実行して Origin リポジトリーからサンプルを使用できます。

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

5. Pipeline を起動します。

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

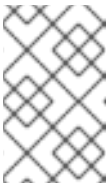
注記

または、OpenShift Container Platform Web コンソールで Builds → Pipeline セクションに移動して、**Start Pipeline** をクリックするか、Jenkins コンソールから作成した Pipeline に移動して、**Build Now** をクリックして Pipeline を起動できます。

パイプラインが起動したら、以下のアクションがプロジェクト内で実行されるはずです。

- ジョブインスタンスが Jenkins サーバー上で作成される
- パイプラインで必要な場合には、エージェント Pod が起動される
- Pipeline がエージェント Pod で実行されるか、エージェントが必要でない場合には master で実行される
  - **template=nodejs-mongodb-example** ラベルの付いた以前に作成されたリソースは削除されます。
  - 新規アプリケーションおよびそれに関連するすべてのリソースは、**nodejs-mongodb-example** テンプレートで作成されます。
  - ビルドは **nodejs-mongodb-example BuildConfig** を使用して起動されます。
    - Pipeline は、ビルドが完了して次のステージをトリガーするまで待機します。
  - デプロイメントは、**nodejs-mongodb-example** のデプロイメント設定を使用して開始されます。
    - パイプラインは、デプロイメントが完了して次のステージをトリガーするまで待機します。
  - ビルドとデプロイに成功すると、**nodejs-mongodb-example:latest** イメージが **nodejs-mongodb-example:stage** としてトリガーされます。

- パイプラインで以前に要求されていた場合には、スレーブ Pod が削除される



注記

OpenShift Container Platform Web コンソールで確認すると、最適な方法で Pipeline の実行を視覚的に把握することができます。Web コンソールにログインして、Builds → Pipelines に移動し、Pipeline を確認します。

2.5.5. Web コンソールを使用したシークレットの追加

プライベートリポジトリーにアクセスできるように、ビルド設定にシークレットを追加することができます。

手順

OpenShift Container Platform Web コンソールからプライベートリポジトリーにアクセスできるようにビルド設定にシークレットを追加するには、以下を実行します。

1. 新規の OpenShift Container Platform プロジェクトを作成します。
2. プライベートのソースコードリポジトリーにアクセスするための認証情報が含まれるシークレットを作成します。
3. ビルド設定を作成します。
4. ビルド設定エディターページまたは Web コンソールの **create app from builder image** ページで、**Source Secret** を設定します。
5. **Save** をクリックします。

2.5.6. プルおよびプッシュの有効化

プライベートレジストリーへのプルを実行できるようにするには、ビルド設定にプルシークレットを設定し、プッシュします。

手順

プライベートレジストリーへのプルを有効にするには、以下を実行します。

- ビルド設定にプルシークレットを設定します。

プッシュを有効にするには、以下を実行します。

- ビルド設定にプッシュシークレットを設定します。

2.6. BUILDAH によるカスタムイメージビルド

OpenShift Container Platform 4.10 では、docker ソケットはホストノードにはありません。これは、カスタムビルドの **mount docker socket** オプションがカスタムビルドイメージ内で使用できる docker ソケットを提供しない可能性がゼロではないことを意味します。

イメージのビルドおよびプッシュにこの機能を必要とする場合、Buildah ツールをカスタムビルドイメージに追加し、これを使用してカスタムビルドロジック内でイメージをビルドし、プッシュします。以下の例は、Buildah でカスタムビルドを実行する方法を示しています。

注記

カスタムビルドストラテジーを使用するためには、デフォルトで標準ユーザーが持たないパーミッションが必要です。このパーミッションはユーザーがクラスターで実行される特権付きコンテナー内で任意のコードを実行することを許可します。このレベルのアクセスを使用するとクラスターが危険にさらされる可能性があるため、このアクセスはクラスターで管理者権限を持つ信頼されたユーザーのみに付与される必要があります。

2.6.1. 前提条件

- カスタムビルドパーミッションを付与する 方法について確認してください。

2.6.2. カスタムビルドアーティファクトの作成

カスタムビルドイメージとして使用する必要のあるイメージを作成する必要があります。

手順

1. 空のディレクトリーからはじめ、以下の内容を含む **Dockerfile** という名前のファイルを作成します。

```
FROM registry.redhat.io/rhel8/buildah
# In this example, `/tmp/build` contains the inputs that build when this
# custom builder image is run. Normally the custom builder image fetches
# this content from some location at build time, by using git clone as an example.
ADD dockerfile.sample /tmp/input/Dockerfile
ADD build.sh /usr/bin
RUN chmod a+x /usr/bin/build.sh
# /usr/bin/build.sh contains the actual custom build logic that will be run when
# this custom builder image is run.
ENTRYPOINT ["/usr/bin/build.sh"]
```

2. 同じディレクトリーに、**dockerfile.sample** という名前のファイルを作成します。このファイルはカスタムビルドイメージに組み込まれ、コンテンツビルドによって生成されるイメージを定義します。

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

3. 同じディレクトリーに、**build.sh** という名前のファイルを作成します。このファイルには、カスタムビルドの実行時に実行されるロジックが含まれます。

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


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

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


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

2.6.3. カスタムビルダーイメージのビルド

OpenShift Container Platform を使用してカスタムストラテジーで使用するカスタムビルダーイメージをビルドし、プッシュすることができます。

前提条件

- 新規カスタムビルダーイメージの作成に使用されるすべての入力を定義します。

手順

1. カスタムビルダーイメージをビルドする **BuildConfig** オブジェクトを定義します。

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

2. カスタムビルドイメージを作成したディレクトリーから、ビルドを実行します。

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

ビルドの完了後に、新規のカスタムビルダーイメージが **custom-builder-image:latest** という名前のイメージストリームタグのプロジェクトで利用可能になります。

2.6.4. カスタムビルダーイメージの使用

カスタムビルダーイメージとカスタムストラテジーを併用する **BuildConfig** オブジェクトを定義し、カスタムビルドロジックを実行することができます。

前提条件

- 新規カスタムビルダーイメージに必要なすべての入力を定義します。
- カスタムビルダーイメージをビルドします。

手順

1. **buildconfig.yaml** という名前のファイルを作成します。このファイルは、プロジェクトに作成され、実行される **BuildConfig** オブジェクトを定義します。

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: sample-custom-build
```

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

1 プロジェクト名を指定します。

2. **BuildConfig** を作成します。

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

3. **imagestream.yaml** という名前のファイルを作成します。このファイルはビルドがイメージをプッシュするイメージストリームを定義します。

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

4. imagestream を作成します。

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

5. カスタムビルドを実行します。

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

ビルドが実行されると、以前にビルドされたカスタムビルダーイメージを実行する Pod が起動します。Pod はカスタムビルダーイメージのエントリーポイントとして定義される **build.sh** ロジックを実行します。**build.sh** ロジックは Buildah を起動し、カスタムビルダーイメージに埋め込まれた **dockerfile.sample** をビルドしてから、Buildah を使用して新規イメージを **sample-custom image stream** にプッシュします。

2.7. 基本的なビルドの実行および設定

以下のセクションでは、ビルドの開始および中止、**BuildConfigs** の編集、**BuildConfig** の削除、ビルドの詳細の表示、およびビルドログへのアクセスを含む基本的なビルド操作についての方法を説明します。

2.7.1. ビルドの開始

現在のプロジェクトに既存のビルド設定から新規ビルドを手動で起動できます。

手順

手動でビルドを開始するには、以下のコマンドを入力します。

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

2.7.1.1. ビルドの再実行

--from-build フラグを使用してビルドを手動で再度実行します。

手順

- 手動でビルドを再実行するには、以下のコマンドを入力します。

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

2.7.1.2. ビルドログのストリーミング

--follow フラグを指定して、**stdout** のビルドのログをストリーミングします。

手順

- **stdout** でビルドのログを手動でストリーミングするには、以下のコマンドを実行します。

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

2.7.1.3. ビルド開始時の環境変数の設定

--env フラグを指定して、ビルドの任意の環境変数を設定します。

手順

- 必要な環境変数を指定するには、以下のコマンドを実行します。

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

2.7.1.4. ソースを使用したビルドの開始

Git ソースプルまたは Dockerfile に依存してビルドするのではなく、ソースを直接プッシュしてビルドを開始することも可能です。ソースには、Git または SVN の作業ディレクトリーの内容、デプロイする事前にビルド済みのバイナリーアーティファクトのセットまたは単一ファイルのいずれかを選択できます。これは、**start-build** コマンドに以下のオプションのいずれかを指定して実行できます。

オプション	説明
--from-dir=<directory>	アーカイブし、ビルドのバイナリー入力として使用するディレクトリーを指定します。

オプション	説明
--from-file=<file>	単一ファイルを指定します。これはビルドソースで唯一のファイルでなければなりません。 このファイルは、元のファイルと同じファイル名で空のディレクトリーのルートに置いてください。
--from-repo=<local_source_repo>	ビルドのバイナリー入力として使用するローカルリポジトリーへのパスを指定します。**--commit** オプションを追加して、ビルドに使用するブランチ、タグ、またはコミットを制御します。

以下のオプションをビルドに直接指定した場合には、コンテンツはビルドにストリーミングされ、現在のビルドソースの設定が上書きされます。

注記

バイナリー入力からトリガーされたビルドは、サーバー上にソースを保存しないため、ベースイメージの変更でビルドが再度トリガーされた場合には、ビルド設定で指定されたソースが使用されます。

手順

- 以下のコマンドを使用してソースからビルドを開始し、タグ **v2** からローカル Git リポジトリーの内容をアーカイブとして送信します。

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

2.7.2. ビルドの中止

Web コンソールまたは以下の CLI コマンドを使用して、ビルドを中止できます。

手順

- 手動でビルドを取り消すには、以下のコマンドを入力します。

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

2.7.2.1. 複数ビルドのキャンセル

以下の CLI コマンドを使用して複数ビルドを中止できます。

手順

- 複数ビルドを手動で取り消すには、以下のコマンドを入力します。

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

2.7.2.2. すべてのビルドのキャンセル

以下の CLI コマンドを使用し、ビルド設定からすべてのビルドを中止できます。

手順

- すべてのビルドを取り消すには、以下のコマンドを実行します。

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

2.7.2.3. 指定された状態のすべてのビルドのキャンセル

特定の状態にあるビルドをすべて取り消すことができます (例: **new** または **pending**)。この際、他の状態のビルドは無視されます。

手順

- 特定の状態のすべてのビルドを取り消すには、以下のコマンドを入力します。

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

2.7.3. BuildConfig の編集

ビルド設定を編集するには、Developer パースペクティブの Builds ビューで Edit BuildConfig オプションを使用します。

以下のいずれかのビューを使用して **BuildConfig** を編集できます。

- Form view を使用すると、標準のフォームフィールドおよびチェックボックスを使用して **BuildConfig** を編集できます。
- YAML ビュー を使用すると、操作を完全に制御して **BuildConfig** を編集できます。

データを失うことなく、Form view と YAML view を切り替えることができます。Form ビュー のデータは YAML ビュー に転送されます (その逆も同様です)。

手順

1. Developer パースペクティブの Builds ビューで、メニュー  をクリックし、Edit BuildConfig オプションを表示します。
2. Edit BuildConfig をクリックし、Form view オプションを表示します。
3. Git セクションで、アプリケーションの作成に使用するコードベースの Git リポジトリー URL を入力します。その後、URL は検証されます。
   - オプション: Show Advanced Git Options をクリックし、以下のような詳細を追加します。
     - Git Reference: アプリケーションのビルドに使用するコードが含まれるブランチ、タグ、またはコミットを指定します。
     - Context Dir: アプリケーションのビルドに使用するアプリケーションのコードが含まれるサブディレクトリーを指定します。
     - Source Secret プライベートリポジトリーからソースコードをプルするための認証情報で Secret Name を作成します。

4. **Build from** セクションで、ビルド元となるオプションを選択します。以下のオプションで使用できます。

   - **イメージストリームタグ** は、所定のイメージストリームおよびタグのイメージを参照します。ビルド元およびプッシュ元の場所に指定するプロジェクト、イメージストリーム、およびタグを入力します。

   - **イメージストリームイメージ** は、所定のイメージストリームのイメージとおよびイメージ名を参照します。ビルドするイメージストリームイメージを入力します。また、プッシュ先となるプロジェクト、イメージストリーム、およびタグも入力します。

   - **Docker image**: Docker イメージは Docker イメージリポジトリーを使用して参照されます。また、プッシュ先の場所を参照するように、プロジェクト、イメージストリーム、タグを入力する必要があります。

5. オプション: **環境変数** セクションで**Name** と **Value** フィールドを使用して、プロジェクトに関連付けられた環境変数を追加します。環境変数を追加するには、**Add Value** または **Add from ConfigMap** と **Secret** を使用します。

6. オプション: 以下の高度なオプションを使用してアプリケーションをさらにカスタマイズできます。

   トリガー

   ビルダーイメージの変更時に新規イメージビルドをトリガーします。**Add Trigger** をクリックし、**Type** および **Secret** を選択して、トリガーを追加します。

   シークレット

   アプリケーションのシークレットを追加します。**Add secret** をクリックし、**Secret** および **Mount point** を選択して、さらにシークレットを追加します。

   Policy

   Run policy をクリックして、ビルド実行ポリシーを選択します。選択したポリシーは、ビルド設定から作成されるビルドを実行する順番を決定します。

   フック

   Run build hooks after image is builtを選択して、ビルドの最後にコマンドを実行し、イメージを検証します。**Hook type**、**Command** および **Arguments** をコマンドに追加しあ m す。

7. **Save** をクリックして **BuildConfig** を保存します。

2.7.4. BuildConfig の削除

以下のコマンドで **BuildConfig** を削除します。

手順

- **BuildConfig** を削除するには、以下のコマンドを入力します。

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

  これにより、この **BuildConfig** でインスタンス化されたビルドがすべて削除されます。

- **BuildConfig** を削除して、 **BuildConfig** からインスタンス化されたビルドを保持するには、以下のコマンドの入力時に **--cascade=false** フラグを指定します。

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

2.7.5. ビルドの詳細表示

Web コンソールまたは **oc describe** CLI コマンドを使用して、ビルドの詳細を表示できます。

これにより、以下のような情報が表示されます。

- ビルドソース
- ビルドストラテジー
- 出力先
- 宛先レジストリーのイメージのダイジェスト
- ビルドの作成方法

ビルドが **Docker** または **Source** ストラテジーを使用する場合、**oc describe** 出力には、コミット ID、作成者、コミットしたユーザー、メッセージなどのビルドに使用するソースのリビジョンの情報が含まれます。

手順

- ビルドの詳細を表示するには、以下のコマンドを入力します。

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

2.7.6. ビルドログへのアクセス

Web コンソールまたは CLI を使用してビルドログにアクセスできます。

手順

- ビルドを直接使用してログをストリーミングするには、以下のコマンドを入力します。

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

2.7.6.1. BuildConfig ログへのアクセス

Web コンソールまたは CLI を使用して **BuildConfig** ログにアクセスできます。

手順

- **BuildConfig** の最新ビルドのログをストリーミングするには、以下のコマンドを入力します。

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

2.7.6.2. 特定バージョンのビルドについての BuildConfig ログへのアクセス

Web コンソールまたは CLI を使用して、**BuildConfig** についての特定バージョンのビルドのログにアクセスすることができます。

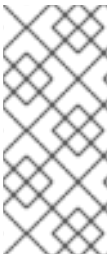
手順

- **BuildConfig** の特定バージョンのビルドのログをストリームするには、以下のコマンドを入力します。

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

2.7.6.3. ログの冗長性の有効化

詳細の出力を有効にするには、**BuildConfig** 内の **sourceStrategy** または **dockerStrategy** の一部として **BUILD_LOGLEVEL** 環境変数を指定します。



注記

管理者は、**env/BUILD_LOGLEVEL** を設定して、OpenShift Container Platform インスタンス全体のデフォルトのビルドの詳細レベルを設定できます。このデフォルトは、指定の **BuildConfig** で **BUILD_LOGLEVEL** を指定することで上書きできます。コマンドラインで **--build-loglevel** を **oc start-build** に渡すことで、バイナリー以外のビルドについて優先順位の高い上書きを指定することができます。

ソースビルドで利用できるログレベルは以下のとおりです。

レベル 0	**assemble** スクリプトを実行してコンテナーからの出力とすべてのエラーを生成します。これはデフォルトになります。
レベル 1	実行したプロセスに関する基本情報を生成します。
レベル 2	実行したプロセスに関する詳細情報を生成します。
レベル 3	実行したプロセスに関する詳細情報と、アーカイブコンテンツのリストを生成します。
レベル 4	現時点ではレベル 3 と同じ情報を生成します。
レベル 5	これまでのレベルで記載したすべての内容と docker のプッシュメッセージを提供します。

手順

- 詳細の出力を有効にするには、**BuildConfig** 内の **sourceStrategy** または **dockerStrategy** の一部として **BUILD_LOGLEVEL** 環境変数を渡します。

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

1 この値を任意のログレベルに調整します。

2.8. ビルドのトリガーおよび変更

以下のセクションでは、ビルドフックを使用してビルドをトリガーし、ビルドを変更する方法についての概要を説明します。

2.8.1. ビルドトリガー

BuildConfig の定義時に、**BuildConfig** を実行する必要のある状況を制御するトリガーを定義できます。以下のビルドトリガーを利用できます。

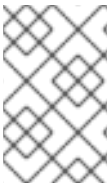
- Webhook
- イメージの変更
- 設定の変更

2.8.1.1. Webhook のトリガー

Webhook のトリガーにより、要求を OpenShift Container Platform API エンドポイントに送信して新規ビルドをトリガーできます。GitHub、GitLab、Bitbucket または Generic webhook を使用してこれらのトリガーを定義できます。

OpenShift Container Platform の Webhook は現在、Git ベースのソースコード管理システム (SCM) システムのそれぞれのプッシュイベントの類似のバージョンのみをサポートしています。その他のイベントタイプはすべて無視されます。

プッシュイベントを処理する場合に、OpenShift Container Platform コントロールプレーンホストは、イベント内のブランチ参照が、対応の **BuildConfig** のブランチ参照と一致しているかどうを確認します。一致する場合には、OpenShift Container Platform ビルドの Webhook イベントに記載されているのと全く同じコミット参照がチェックアウトされます。一致しない場合には、ビルドはトリガーされません。



注記

oc new-app および **oc new-build** は GitHub および Generic Webhook トリガーを自動的に作成しますが、それ以外の Webhook トリガーが必要な場合には手動で追加する必要があります。トリガーを設定して、トリガーを手動で追加できます。

Webhook すべてに対して、**WebHookSecretKey** という名前のキーでシークレットと、Webook の呼び出し時に提供される値を定義する必要があります。webhook の定義で、このシークレットを参照する必要があります。このシークレットを使用することで URL が一意となり、他の URL でビルドがトリガーされないようにします。キーの値は、webhook の呼び出し時に渡されるシークレットと比較されます。

たとえば、**mysecret** という名前のシークレットを参照する GitHub webhook は以下のとおりです。

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

次に、シークレットは以下のように定義します。シークレットの値は base64 エンコードされており、この値は **Secret** オブジェクトの **data** フィールドに必要である点に注意してください。

```
- kind: Secret
  apiVersion: v1
  metadata:
    name: mysecret
```

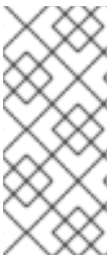
```
    creationTimestamp:
  data:
    WebHookSecretKey: c2VjcmV0dmFsdWUx
```

2.8.1.1.1. GitHub Webhook の使用

GitHub webhook は、リポジトリーの更新時に GitHub からの呼び出しを処理します。トリガーを定義する際に、シークレットを指定する必要があります。このシークレットは、Webhook の設定時に GitHub に指定する URL に追加されます。

GitHub Webhook の定義例:

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



注記

Webhook トリガーの設定で使用されるシークレットは、GitHub UI で Webhook の設定時に表示される **secret** フィールドとは異なります。Webhook トリガー設定で使用するシークレットは、Webhook URL を一意にして推測ができないようにし、GitHub UI のシークレットは、任意の文字列フィールドで、このフィールドを使用して本体の HMAC hex ダイジェストを作成して、**X-Hub-Signature** ヘッダーとして送信します。

oc describe コマンドは、ペイロード URL を GitHub Webhook URL として返します (Webhook URL の表示を参照)。 ペイロード URL は以下のように設定されます。

出力例

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

前提条件

- GitHub リポジトリーから **BuildConfig** を作成します。

手順

1. GitHub Webhook を設定するには以下を実行します。
   a. GitHub リポジトリーから **BuildConfig** を作成した後に、以下を実行します。

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

      以下のように、上記のコマンドは Webhook GitHub URL を生成します。

      出力例

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

b. GitHub の Web コンソールから、この URL を GitHub にカットアンドペーストします。

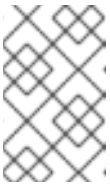
c. GitHub リポジトリーで、**Settings → Webhooks** から **Add Webhook** を選択します。

d. **Payload URL** フィールドに、URL の出力を貼り付けます。

e. **Content Type** を GitHub のデフォルト **application/x-www-form-urlencoded** から **application/json** に変更します。

f. **Add webhook** をクリックします。
webhook の設定が正常に完了したことを示す GitHub のメッセージが表示されます。

これで変更を GitHub リポジトリーにプッシュする際に新しいビルドが自動的に起動し、ビルドに成功すると新しいデプロイメントが起動します。



注記

Gogs は、GitHub と同じ webhook のペイロード形式をサポートします。そのため、Gogs サーバーを使用する場合は、GitHub webhook トリガーを **BuildConfig** に定義すると、Gogs サーバー経由でもトリガーされます。

2. **payload.json** などの有効な JSON ペイロードがファイルに含まれる場合には、**curl** を使用して webhook を手動でトリガーできます。

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

-k の引数は、API サーバーに正しく署名された証明書がない場合にのみ必要です。

注記

ビルドは、GitHub Webhook イベントからの **ref** 値が、**BuildConfig** リソースの **source.git** フィールドで指定された **ref** 値と一致する場合にのみトリガーされます。

関連情報

- Gogs

2.8.1.1.2. GitLab Webhook の使用

GitLab Webhook は、リポジトリーの更新時の GitLab による呼び出しを処理します。GitHub トリガーでは、シークレットを指定する必要があります。以下の例は、**BuildConfig** 内のトリガー定義の YAML です。

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

oc describe コマンドは、ペイロード URL を GitLab Webhook URL として返します。 ペイロード URL は以下のように設定されます。

出力例

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

手順

1. GitLab Webhook を設定するには以下を実行します。
   a. **BuildConfig** を Webhook URL を取得するように記述します。

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

   b. Webhook URL をコピーします。 **<secret>** はシークレットの値に置き換えます。
   c. GitLab の設定手順 に従い、GitLab リポジトリーの設定に Webhook URL を貼り付けます。
2. **payload.json** などの有効な JSON ペイロードがファイルに含まれる場合には、 **curl** を使用して webhook を手動でトリガーできます。

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

   -k の引数は、API サーバーに正しく署名された証明書がない場合にのみ必要です。

2.8.1.1.3. Bitbucket Webhook の使用

Bitbucket webhook は、リポジトリーの更新時の Bitbucket による呼び出しを処理します。これまでのトリガーと同様に、シークレットを指定する必要があります。以下の例は、**BuildConfig** 内のトリガー定義の YAML です。

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

oc describe コマンドは、ペイロード URL を Bitbucket Webhook URL として返します。ペイロード URL は以下のように設定されます。

出力例

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

手順

1. Bitbucket Webhook を設定するには以下を実行します。
   a. 'BuildConfig' を記述して Webhook URL を取得します。

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

b. Webhook URL をコピーします。 **<secret>** はシークレットの値に置き換えます。

c. Bitbucket の設定手順 に従い、Bitbucket リポジトリーの設定に Webhook URL を貼り付けます。

2. **payload.json** などの有効な JSON ペイロードがファイルに含まれる場合には、 **curl** を使用して webhook を手動でトリガーできます。

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

-k の引数は、API サーバーに正しく署名された証明書がない場合にのみ必要です。

2.8.1.1.4. Generic Webhook の使用

Generic Webhook は、Web 要求を実行できるシステムから呼び出されます。他の webhook と同様に、シークレットを指定する必要があります。このシークレットは、呼び出し元がビルドをトリガーするために使用する必要のある URL に追加されます。このシークレットを使用することで URL が一意となり、他の URL でビルドがトリガーされないようにします。以下の例は、**BuildConfig** 内のトリガー定義の YAML です。

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

1 **true** に設定して、Generic Webhook が環境変数で渡させるようにします。

手順

1. 呼び出し元を設定するには、呼び出しシステムに、ビルドの Generic Webhook エンドポイントの URL を指定します。

出力例

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

呼び出し元は、**POST** 操作として Webhook を呼び出す必要があります。

2. 手動で Webhook を呼び出すには、**curl** を使用します。

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

HTTP 動詞は **POST** に設定する必要があります。セキュアでない **-k** フラグを指定して、証明書の検証を無視します。クラスターに正しく署名された証明書がある場合には、2 つ目のフラグは必要ありません。

エンドポイントは、以下の形式で任意のペイロードを受け入れることができます。

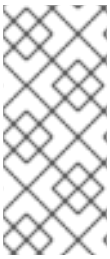
```
git:
  uri: "<url to git repository>"
  ref: "<optional git reference>"
  commit: "<commit hash identifying a specific git commit>"
  author:
    name: "<author name>"
    email: "<author e-mail>"
  committer:
    name: "<committer name>"
    email: "<committer e-mail>"
  message: "<commit message>"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

(1) **BuildConfig** 環境変数と同様に、ここで定義されている環境変数はビルドで利用できます。これらの変数が **BuildConfig** の環境変数と競合する場合には、これらの変数が優先されます。デフォルトでは、webhook 経由で渡された環境変数は無視されます。Webhook 定義の **allowEnv** フィールドを **true** に設定して、この動作を有効にします。

3. **curl** を使用してこのペイロードを渡すには、**payload_file.yaml** という名前のファイルにペイロードを定義して実行します。

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

引数は、ヘッダーとペイロードを追加した以前の例と同じです。**-H** の引数は、ペイロードの形式により **Content-Type** ヘッダーを **application/yaml** または **application/json** に設定します。**--data-binary** の引数を使用すると、**POST** 要求では、改行を削除せずにバイナリーペイロードを送信します。



注記

OpenShift Container Platform は、要求のペイロードが無効な場合でも (例: 無効なコンテンツタイプ、解析不可能または無効なコンテンツなど)、Generic Webhook 経由でビルドをトリガーできます。この動作は、後方互換性を確保するために継続されています。無効な要求ペイロードがある場合には、OpenShift Container Platform は、**HTTP 200 OK** 応答の一部として JSON 形式で警告を返します。

2.8.1.1.5. Webhook URL の表示

以下のコマンドを使用して、ビルド設定に関連する webhook URL を表示できます。コマンドが Webhook URL を表示しない場合、そのビルド設定に定義される Webhook トリガーはありません。

手順

- **BuildConfig** に関連付けられた Webhook URL を表示するには、以下を実行します。

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

2.8.1.2. イメージ変更トリガーの使用

開発者は、ベースイメージが変更するたびにビルドを自動的に実行するように設定できます。

イメージ変更のトリガーを使用すると、アップストリームイメージで新規バージョンが利用できるようになると、ビルドが自動的に呼び出されます。たとえば、RHEL イメージ上にビルドが設定されている場合に、RHEL のイメージが変更された時点でビルドの実行をトリガーできます。その結果、アプリケーションイメージは常に最新の RHEL ベースイメージ上で実行されるようになります。

注記

v1 コンテナーレジストリー のコンテナーイメージを参照するイメージストリームは、イメージストリームタグが利用できるようになった時点でビルドが 1 度だけトリガーされ、後続のイメージ更新ではトリガーされません。これは、v1 コンテナーレジストリーに一意で識別可能なイメージがないためです。

手順

1. トリガーするアップストリームイメージを参照するように、**ImageStream** を定義します。

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

この定義では、イメージストリームが **<system-registry>/<namespace>/ruby-20-centos7** に配置されているコンテナーイメージリポジトリーに紐付けられます。**<system-registry>** は、OpenShift Container Platform で実行する名前が **docker-registry** のサービスとして定義されます。

2. イメージストリームがビルドのベースイメージの場合には、ビルドストラテジーの **from** フィールドを設定して、**ImageStream** を参照します。

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

上記の例では、**sourceStrategy** の定義は、この namespace 内に配置されている **ruby-20-centos7** という名前のイメージストリームの **latest** タグを使用します。

3. **ImageStreams** を参照する 1 つまたは複数のトリガーでビルドを定義します。

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

1. ビルドストラテジーの **from** フィールドに定義されたように **ImageStream** および **Tag** を監視するイメージ変更トリガー。ここの **imageChange** オブジェクトは空でなければなり

2. 任意のイメージストリームを監視するイメージ変更トリガー。この例に含まれる **imageChange** の部分には **from** フィールドを追加して、監視する **ImageStreamTag** を参照させる必要があります。

ストラテジーイメージストリームにイメージ変更トリガーを使用する場合は、生成されたビルドに不変な docker タグが付けられ、そのタグに対応する最新のイメージを参照させます。この新規イメージ参照は、ビルド用に実行するときに、ストラテジーにより使用されます。

ストラテジーイメージストリームを参照しない、他のイメージ変更トリガーの場合は、新規ビルドが開始されますが、一意のイメージ参照で、ビルドストラテジーは更新されません。

この例には、ストラテジーについてのイメージ変更トリガーがあるので、結果として生成されるビルドは以下のようになります。

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

これにより、トリガーされたビルドは、リポジトリーにプッシュされたばかりの新しいイメージを使用して、ビルドが同じ入力内容でいつでも再実行できるようにします。

参照されるイメージストリームで複数の変更を可能にするためにイメージ変更トリガーを一時停止してからビルドを開始できます。また、ビルドがすぐにトリガーされるのを防ぐために、最初に **ImageChangeTrigger** を **BuildConfig** に追加する際に、**paused** 属性を true に設定することもできます。

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

カスタムビルドの場合、すべての **Strategy** タイプにイメージフィールドを設定するだけでなく、**OPENSHIFT_CUSTOM_BUILD_BASE_IMAGE** の環境変数もチェックされます。この環境変数が存在しない場合は、不変のイメージ参照で作成されます。存在する場合には、この不変のイメージ参照で更新されます。

ビルドが Webhook トリガーまたは手動の要求でトリガーされた場合に、作成されるビルドは、**Strategy** が参照する **ImageStream** から解決する **<immutableid>** を使用します。これにより、簡単に再現できるように、一貫性のあるイメージタグを使用してビルドが実行されるようになります。

関連情報

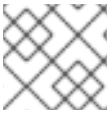
- v1 コンテナーレジストリー

2.8.1.3. ビルドのイメージ変更トリガーの識別

開発者は、イメージ変更トリガーがある場合は、どのイメージの変更が最後のビルドを開始したかを特定できます。これは、ビルドのデバッグやトラブルシューティングに役立ちます。

BuildConfig の例

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

# ...

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

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

  lastVersion: 1
```



注記

この例では、イメージ変更トリガーに関係のない要素を省略します。

前提条件

- 複数のイメージ変更トリガーを設定している。これらのトリガーは 1 つまたは複数のビルドがトリガーされています。

手順

1. **buildConfig.status.imageChangeTriggers** で、最新のタイムスタンプを持つ **lastTriggerTime** を特定します。
   This **ImageChangeTriggerStatus**

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

2. **imageChangeTriggers** でタイムスタンプを比較して最新のものを特定します。

イメージ変更のトリガー

ビルド設定で、**buildConfig.spec.triggers** はビルドトリガーポリシー (**BuildTriggerPolicy**) の配列です。

各 **BuildTriggerPolicy** には **type** フィールドと、ポインターフィールドのセットがあります。各ポインターフィールドは、**type** フィールドに許可される値の 1 つに対応します。そのため、**BuildTriggerPolicy** を 1 つのポインターフィールドのみに設定できます。

イメージ変更のトリガーの場合、**type** の値は **ImageChange** です。次に、**imageChange** フィールドは、以下のフィールドを持つ **ImageChangeTrigger** オブジェクトへのポインターです。

- **lastTriggeredImageID**: このフィールドは例では提供されず、OpenShift Container Platform 4.8 で非推奨となり、今後のリリースでは無視されます。これには、最後のビルドがこの **BuildConfig** からトリガーされた際に **ImageStreamTag** の解決されたイメージ参照が含まれます。

- **paused**: このフィールドは、この例では示されていませんが、この特定のイメージ変更トリガーを一時的に無効にするのに使用できます。

- **from**: このフィールドを使用して、このイメージ変更トリガーを駆動する **ImageStreamTag** を参照します。このタイプは、コア Kubernetes タイプである **OwnerReference** です。

from フィールドには、注意フィールド **kind があります。イメージ変更トリガーの場合、サポートされる値は ImageStreamTag のみです。 namespace**: このフィールドを使用して **ImageStreamTag** の namespace を指定します。** **name**: このフィールドを使用して **ImageStreamTag** を指定します。

イメージ変更のトリガーのステータス

ビルド設定で、**buildConfig.status.imageChangeTriggers** は **ImageChangeTriggerStatus** 要素の配列です。それぞれの **ImageChangeTriggerStatus** 要素には、前述の例に示されている **from**、**lastTriggeredImageID**、および **lastTriggerTime** 要素が含まれます。

最新の **lastTriggerTime** を持つ **ImageChangeTriggerStatus** は、最新のビルドをトリガーしました。**name** および **namespace** を使用して、ビルドをトリガーした **buildConfig.spec.triggers** でイメージ変更トリガーを特定します。

lastTriggerTime は最新のタイムスタンプ記号で、最後のビルドの **ImageChangeTriggerStatus** を示します。この **ImageChangeTriggerStatus** には、ビルドをトリガーした **buildConfig.spec.triggers** のイメージ変更トリガーと同じ **name** および **namespace** があります。

関連情報

- v1 コンテナーレジストリー

2.8.1.4. 設定変更のトリガー

設定変更トリガーにより、新規の **BuildConfig** が作成されるとすぐに、ビルドが自動的に起動されます。

以下の例は、**BuildConfig** 内のトリガー定義の YAML です。

```
type: "ConfigChange"
```

注記

設定変更のトリガーは新しい **BuildConfig** が作成された場合のみ機能します。今後のリリースでは、設定変更トリガーは、**BuildConfig** が更新されるたびにビルドを起動できるようになります。

2.8.1.4.1. トリガーの手動設定

トリガーは、**oc set triggers** を使用してビルド設定に対して追加/削除できます。

手順

- ビルド設定に GitHub Webhook トリガーを設定するには、以下を使用します。

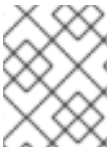
```
$ oc set triggers bc <name> --from-github
```

- イメージ変更トリガーを設定するには、以下を使用します。

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

- トリガーを削除するには **--remove** を追加します。

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



注記

Webhook トリガーがすでに存在する場合には、トリガーをもう一度追加すると、Webhook のシークレットが再生成されます。

詳細情報は、以下を実行してヘルプドキュメントを参照してください。

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

2.8.2. ビルドフック

ビルドフックを使用すると、ビルドプロセスに動作を挿入できます。

BuildConfig オブジェクトの **postCommit** フィールドにより、ビルドアウトプットイメージを実行する一時的なコンテナー内でコマンドが実行されます。イメージの最後の層がコミットされた直後、かつイメージがレジストリーにプッシュされる前に、フックが実行されます。

現在の作業ディレクトリーは、イメージの **WORKDIR** に設定され、コンテナーイメージのデフォルトの作業ディレクトリーになります。多くのイメージでは、ここにソースコードが配置されます。

ゼロ以外の終了コードが返された場合、一時コンテナーの起動に失敗した場合には、フックが失敗します。フックが失敗すると、ビルドに失敗とマークされ、このイメージはレジストリーにプッシュされません。失敗の理由は、ビルドログを参照して検証できます。

ビルドフックは、ビルドが完了とマークされ、イメージがレジストリーに公開される前に、単体テストを実行してイメージを検証するために使用できます。すべてのテストに合格し、テストランナーにより終了コード **0** が返されると、ビルドは成功とマークされます。テストに失敗すると、ビルドは失敗とマークされます。すべての場合に、ビルドログにはテストランナーの出力が含まれるので、失敗したテストを特定するのに使用できます。

postCommit フックは、テストの実行だけでなく、他のコマンドにも使用できます。一時的なコンテナーで実行されるので、フックによる変更は永続されず、フックの実行は最終的なイメージには影響がありません。この動作はさまざまな用途がありますが、これにより、テストの依存関係がインストール、使用されて、自動的に破棄され、最終イメージには残らないようにすることができます。

2.8.2.1. コミット後のビルドフックの設定

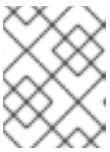
ビルド後のフックを設定する方法は複数あります。以下の例に出てくるすべての形式は同等で、**bundle exec rake test --verbose** を実行します。

手順

- シェルスクリプト:

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

script の値は、**/bin/sh -ic** で実行するシェルスクリプトです。上記のように単体テストを実行する場合など、シェルスクリプトがビルドフックの実行に適している場合に、これを使用します。たとえば、上記のユニットテストを実行する場合などです。イメージのエントリーポイントを制御するか、イメージに **/bin/sh** がない場合は、**command** および/または **args** を使用します。



注記

CentOS や RHEL イメージでの作業を改善するために、追加で **-i** フラグが導入されましたが、今後のリリースで削除される可能性があります。

- イメージエントリーポイントとしてのコマンド:

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

この形式では **command** は実行するコマンドで、Dockerfile 参照 に記載されている、実行形式のイメージエントリーポイントを上書きします。Command は、イメージに **/bin/sh** がない、またはシェルを使用しない場合に必要です。他の場合は、**script** を使用することが便利な方法になります。

- 引数のあるコマンド:

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

この形式は **command** に引数を追加するのと同じです。

注記

script と **command** を同時に指定すると、無効なビルドフックが作成されてしまいます。

2.8.2.2. CLI を使用したコミット後のビルドフックの設定

oc set build-hook コマンドを使用して、ビルド設定のビルドフックを設定することができます。

手順

1. コミット後のビルドフックとしてコマンドを設定します。

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

2. コミット後のビルドフックとしてスクリプトを設定します。

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

2.9. 高度なビルドの実行

以下のセクションでは、ビルドリソースおよび最長期間の設定、ビルドのノードへの割り当て、チェーンビルド、ビルドのプルーニング、およびビルド実行ポリシーなどの高度なビルド操作について説明します。

2.9.1. ビルドリソースの設定

デフォルトでは、ビルドは、メモリーや CPU など、バインドされていないリソースを使用して Pod により完了されます。これらのリソースは制限できます。

手順

リソースの使用を制限する方法は 2 つあります。

- プロジェクトのデフォルトコンテナー制限でリソース制限を指定して、リソースを制限します。
- リソースの制限をビルド設定の一部として指定し、リソースの使用を制限します。** 以下の例では、**resources**、 **cpu**、および **memory** パラメーターはそれぞれオプションです。

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

-

1. **cpu** は CPU のユニットで、**100m** は 0.1 CPU ユニット (100 * 1e-3) を表します。
2. **memory** はバイト単位です。**256Mi** は 268435456 バイトを表します (256 * 2 ^ 20)。

ただし、クォータがプロジェクトに定義されている場合には、以下の 2 つの項目のいずれかが必要です。

- 明示的な **requests** で設定した **resources** セクション:

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

1. **requests** オブジェクトは、クォータ内のリソースリストに対応するリソースリストを含みます。

- プロジェクトに定義される制限範囲。 **LimitRange** オブジェクトからのデフォルト値がビルドプロセス時に作成される Pod に適用されます。
  適用されない場合は、クォータ基準を満たさないために失敗したというメッセージが出され、ビルド Pod の作成は失敗します。

2.9.2. 最長期間の設定

BuildConfig オブジェクトの定義時に、**completionDeadlineSeconds** フィールドを設定して最長期間を定義できます。このフィールドは秒単位で指定し、デフォルトでは設定されません。設定されていない場合は、最長期間は有効ではありません。

最長期間はビルドの Pod がシステムにスケジュールされた時点から計算され、ビルダーイメージをプルするのに必要な時間など、ジョブが有効である期間を定義します。指定したタイムアウトに達すると、ジョブは OpenShift Container Platform により終了されます。

手順

- 最長期間を設定するには、**BuildConfig** に **completionDeadlineSeconds** を指定します。以下の例は **BuildConfig** の一部で、**completionDeadlineSeconds** フィールドを 30 分に指定しています。

```
spec:
  completionDeadlineSeconds: 1800
```

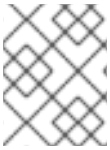
注記

この設定は、パイプラインストラテジーオプションではサポートされていません。

2.9.3. 特定のノードへのビルドの割り当て

ビルドは、ビルド設定の **nodeSelector** フィールドにラベルを指定して、特定のノード上で実行するようにターゲットを設定できます。**nodeSelector** の値は、ビルド Pod のスケジュール時の **Node** ラベルに一致するキー/値のペアに指定してください。

nodeSelector の値は、クラスター全体のデフォルトでも制御でき、値を上書きできます。ビルド設定で **nodeSelector** のキー/値ペアが定義されておらず、 **nodeSelector:{}** が明示的に空になるように定義されていない場合にのみ、デフォルト値が適用されます。値を上書きすると、キーごとにビルド設定の値が置き換えられます。



注記

指定の **NodeSelector** がこれらのラベルが指定されているノードに一致しない場合には、ビルドは **Pending** の状態が無限に続きます。

手順

- 以下のように、**BuildConfig** の **nodeSelector** フィールドにラベルを割り当て、特定の一度で実行されるビルドを割り当てます。

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

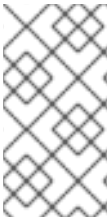
1 このビルド設定に関連するビルドは、**key1=value2** と **key2=value2** ラベルが指定されたノードでのみ実行されます。

2.9.4. チェーンビルド

コンパイル言語 (Go、C、C++、Java など) の場合には、アプリケーションイメージにコンパイルに必要な依存関係を追加すると、イメージのサイズが増加したり、悪用される可能性のある脆弱性が発生したりする可能性があります。

これらの問題を回避するには、2 つのビルドをチェーンでつなげることができます。1つ目のビルドでコンパイルしたアーティファクトを作成し、2 つ目のビルドで、アーティファクトを実行する別のイメージにそのアーティファクトを配置します。

以下の例では、Source-to-Image (S2I) ビルドが docker ビルドに組み合わされ、別のランタイムイメージに配置されるアーティファクトがコンパイルされます。



注記

この例では、S2I ビルドと docker ビルドをチェーンでつないでいますが、1つ目のビルドは、必要なアーティファクトを含むイメージを生成するストラテジーを使用し、2 つ目のビルドは、イメージからの入力コンテンツを使用できるストラテジーを使用できます。

最初のビルドは、アプリケーションソースを取得して、**WAR** ファイルを含むイメージを作成します。このイメージは、**artifact-image** イメージストリームにプッシュされます。アウトプットアーティファクトのパスは、使用する S2I ビルダーの **assemble** スクリプトにより異なります。この場合、**/wildfly/standalone/deployments/ROOT.war** に出力されます。

```
apiVersion: build.openshift.io/v1
```

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

2 つ目のビルドは、1 つ目のビルドからのアウトプットイメージ内にある WAR ファイルへのパスが指定されているイメージソースを使用します。インライン **dockerfile** は、 **WAR** ファイルをランタイムイメージにコピーします。

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: image-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: image-build:latest
  source:
    dockerfile: |-
      FROM jee-runtime:latest
      COPY ROOT.war /deployments/ROOT.war
    images:
    - from: 1
        kind: ImageStreamTag
        name: artifact-image:latest
      paths: 2
      - sourcePath: /wildfly/standalone/deployments/ROOT.war
        destinationDir: "."
  strategy:
    dockerStrategy:
      from: 3
        kind: ImageStreamTag
        name: jee-runtime:latest
  triggers:
  - imageChange: {}
    type: ImageChange
```

1 **from** は、docker ビルドに、以前のビルドのターゲットであった **artifact-image** イメージストリームからのイメージの出力を追加する必要があることを指定します。

2 **paths** は、現在の docker ビルドに追加するターゲットイメージからのパスを指定します。

3 ランタイムのイメージは、docker ビルドのソースイメージとして使用します。

この設定の結果、2 番目のビルドのアウトプットイメージに、**WAR** ファイルの作成に必要なビルドツールを含める必要がなくなります。また、この 2 番目のビルドにはイメージ変更のトリガーが含まれているので、1 番目のビルドがバイナリーアーティファクトで新規イメージを実行して作成するたびに、2 番目のビルドが自動的に、そのアーティファクトを含むランタイムイメージを生成するためにトリガーされます。そのため、どちらのビルドも、ステージが 2 つある単一ビルドのように振る舞います。

2.9.5. ビルドのプルーニング

デフォルトで、ライフサイクルを完了したビルドは無制限に保持されます。保持される以前のビルドの数を制限することができます。

手順

1. **successfulBuildsHistoryLimit** または **failedBuildsHistoryLimit** の正の値を **BuildConfig** に指定して、保持される以前のビルドの数を制限します。 以下は例になります。

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

1 **successfulBuildsHistoryLimit** は、**completed** のステータスのビルドを最大 2 つまで保持します。

2 **failedBuildsHistoryLimit** はステータスが **failed**、**cancelled** または **error** のビルドを最大 2 つまで保持します。

2. 以下の動作のいずれかを実行して、ビルドのプルーニングをトリガーします。
   - ビルド設定が更新された場合
   - ビルドがそのライフサイクルを完了するのを待機します。

ビルドは、作成時のタイムスタンプで分類され、一番古いビルドが先にプルーニングされます。

注記

管理者は、'oc adm' オブジェクトプルーニングコマンドを使用して、ビルドを手動でプルーニングできます。

2.9.6. ビルド実行ポリシー

ビルド実行ポリシーでは、ビルド設定から作成されるビルドを実行する順番を記述します。これには、**Build** の **spec** セクションにある **runPolicy** フィールドの値を変更してください。

既存のビルド設定の **runPolicy** 値を変更することも可能です。以下を実行します。

- **Parallel** から **Serial** や **SerialLatestOnly** に変更して、この設定から新規ビルドをトリガーすると、新しいビルドは並列ビルドすべてが完了するまで待機します。 これは、順次ビルドは、一度に1つしか実行できないためです。
- **Serial** を **SerialLatestOnly** に変更して、新規ビルドをトリガーすると、現在実行中のビルドと直近で作成されたビルド以外には、キューにある既存のビルドがすべてキャンセルされます。最新のビルドが次に実行されます。

2.10. ビルドでの RED HAT サブスクリプションの使用

以下のセクションを使用して、OpenShift Container Platform でエンタイトルメントが適用されたビルドを実行します。

2.10.1. Red Hat Universal Base Image へのイメージストリームタグの作成

ビルド内で Red Hat サブスクリプションを使用するには、Universal Base Image (UBI) を参照するイメージストリームを作成します。

UBI をクラスター内の **すべてのプロジェクトで** 利用可能にするには、イメージストリームタグを **openshift** namespace に追加します。それ以外の場合は、これを **特定のプロジェクトで** 利用可能にするには、イメージストリームタグをそのプロジェクトに追加します。

このようにイメージストリームタグを使用すると、他のユーザーにプルシークレットを公開せずに、インストールプルシークレットの **registry.redhat.io** 認証情報に基づいて UBI へのアクセスを付与することができます。これは、各開発者が各プロジェクトで **registry.redhat.io** 認証情報を使用してプルシークレットをインストールすることが必要になる場合よりも便利です。

手順

- **openshift** namespace で **ImageStreamTag** を作成し、これを開発者に対してすべてのプロジェクトで利用可能にするには、以下を実行します。

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

ヒント

または、以下の YAML を適用して **openshift** namespace に **ImageStreamTag** を作成できます。

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

- 単一プロジェクトで **ImageStreamTag** を作成するには、以下を実行します。

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

ヒント

または、以下の YAML を適用して単一のプロジェクトに **ImageStreamTag** を作成できます。

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

2.10.2. ビルドシークレットとしてのサブスクリプションエンタイトルメントの追加

Red Hat サブスクリプションを使用してコンテンツをインストールするビルドには、ビルドシークレットとしてエンタイトルメントキーを含める必要があります。

前提条件

サブスクリプションを使用して Red Hat エンタイトルメントにアクセスできる。エンタイトルメントシークレットは Insights Operator によって自動的に作成されます。

ヒント

Red Hat Enterprise Linux (RHEL) 7 を使用してエンタイトルメントビルドを実行する場合、**yum** コマンドを実行する前に、Dockerfile に次の手順を含める必要があります。

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

手順

1. etc-pki-entitlement シークレットをビルド設定の Docker ストラテジーでビルドボリュームとして追加します。

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

```
      type: Secret
      secret:
        secretName: etc-pki-entitlement
```

2.10.3. Subscription Manager を使用したビルドの実行

2.10.3.1. Subscription Manager を使用した Docker ビルド

Docker ストラテジービルドは Subscription Manager を使用してサブスクリプションコンテンツをインストールできます。

前提条件

エンタイトルメントキーは、ビルドストラテジーのボリュームとして追加する必要があります。

手順

以下を Dockerfile の例として使用し、Subscription Manager でコンテンツをインストールします。

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

2.10.4. Red Hat Satellite サブスクリプションを使用したビルドの実行

2.10.4.1. Red Hat Satellite 設定のビルドへの追加

Red Hat Satellite を使用してコンテンツをインストールするビルドは、Satellite リポジトリーからコンテンツを取得するための適切な設定を提供する必要があります。

前提条件

- Satellite インスタンスからコンテンツをダウンロードするために、**yum** 互換リポジトリー設定ファイルを提供するか、これを作成する必要があります。

  サンプルリポジトリーの設定

  ```
  [test-<name>]
  name=test-<number>
  baseurl = https://satellite.../content/dist/rhel/server/7/7Server/x86_64/os
  enabled=1
  gpgcheck=0
  sslverify=0
  sslclientkey = /etc/pki/entitlement/...-key.pem
  sslclientcert = /etc/pki/entitlement/....pem
  ```

手順

1. Satellite リポジトリーの設定ファイルを含む **ConfigMap** を作成します。

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

2. Satellite リポジトリー設定およびエンタイトルメントキーをビルドボリュームとして追加します。

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

2.10.4.2. Red Hat Satellite サブスクリプションを使用した Docker ビルド

Docker ストラテジービルドは、Red Hat Satellite リポジトリーを使用してサブスクリプションコンテンツをインストールできます。

前提条件

- エンタイトルメントキーと Satellite リポジトリー設定がビルドボリュームとして追加しておく。

手順

以下のサンプル Dockerfile を使用して、Satellite を使用してコンテンツをインストールします。

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

関連情報

- Red Hat Satellite サブスクリプションと使用する証明書でビルドを使用する方法

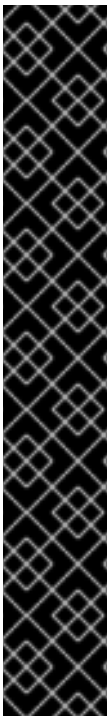
2.10.5. SharedSecret オブジェクトを使用したエンタイトルメントが適用されたビルドの実行

別の namespace の**Secret**オブジェクトからの RHEL エンタイトルメントを安全に使用する 1 つの namespace で、ビルドを設定および実行できます。

Buildオブジェクトと同じ namespace にサブスクリプションクレデンシャルを使用して**Secret**オブジェクトを作成することにより、OpenShift Builds から RHEL エンタイトルメントに引き続きアクセスできます。ただし、OpenShift Container Platform 4.10 以降では、OpenShift Container Platform システム

namespace の 1 つにあ る**Secret** オブジェクトから、クレデンシャルと証明書にアクセスできるようになりました。**Secret** オブジェクトを参照する **SharedSecret** カスタムリソース (CR) インスタンスの CSI ボリュームマウントを使用して、エンタイトルメントのあるビルドを実行します。

この手順は、新しく導入された共有リソース CSI ドライバー機能に依存しています。この機能を使用して、OpenShift Container Platform Builds で CSI ボリュームマウントを宣言できます。これは、OpenShift Container Platform Insights Operator にも依存しています。



重要

共有リソース CSI ドライバーとビルド CSI ボリュームはどちらもテクノロジープレビュー機能であり、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、機能的に完全ではない可能性があります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

共有リソース CSI ドライバーおよびビルド CSI ボリューム機能も、現在のテクノロジープレビュー機能のサブセットである **TechPreviewNoUpgrade** 機能セットに属しています。テストクラスターで **TechPreviewNoUpgrade** 機能セットを有効にできます。この場合、実稼働クラスターで機能を無効にしたまま、完全にテストできます。この機能セットを有効にすると元に戻すことができなくなり、マイナーバージョン更新ができなくなります。この機能セットは、実稼働クラスターでは推奨されません。以下の関連情報セクションの "Enabling Technology Preview features using feature gates" を参照してください。

前提条件

- 機能ゲートを使用して、**TechPreviewNoUpgrade** 機能セットを有効にしている。
- Insights Operator がサブスクリプションクレデンシャルを格納する **Secret** オブジェクトを参照する **SharedSecret** カスタムリソース (CR) インスタンスがある。
- 次のアクションを実行するためのパーミッションがある。
  - ビルド設定を作成し、ビルドを開始します。
  - **oc get sharedsecrets**コマンドを入力し、空でないリストを取得して、使用可能な**SharedSecret** CR インスタンスを見つけます。
  - namespace で使用可能な **builder** サービスアカウントが、指定された **SharedSecret** CR インスタンスの使用を許可されているかどうかを確認します。つまり、**oc adm policy who-can use <identifier of specific SharedSecret>** を使用して、namespace の**builder**サービスアカウントが一覧表示されているかどうかを確認できます。

注記

このリストの最後の 2 つの前提条件のいずれも満たされない場合は、必要なロールベースアクセス制御 (RBAC) を自身で確立するか、誰かに依頼して確立します。これにより、**SharedSecret** CR インスタンスを検出し、サービスアカウントを有効にして **SharedSecret** CR インスタンスを使用できるようになります。

手順

1. YAML コンテンツで**oc apply**を使用して、**SharedSecret** CR インスタンスを使用するための **builder** サービスアカウント RBAC 権限を付与します。

注記

現在、 **kubectl**と**oc**には、**use** 動詞を Pod セキュリティーを中心としたロールに制限する特別な場合のロジックがハードコーディングされています。したがって、**oc create role ...** を使用して、**SharedSecret** CR インスタンスの使用に必要なロールを作成することはできません。

YAML Role オブジェクト定義を使用した oc apply -f コマンドの例

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

2. **oc** コマンドを使用して、ロールに関連付けられた **RoleBinding** を作成します。

oc create rolebinding コマンドの例

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

3. RHEL エンタイトルメントにアクセスする **BuildConfig** オブジェクトを作成します。

YAML BuildConfig オブジェクト定義の例

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: my-csi-bc
  namespace: my-csi-app-namespace
spec:
  runPolicy: Serial
  source:
    dockerfile: |
      FROM registry.redhat.io/ubi8/ubi:latest
      RUN ls -la /etc/pki/entitlement
      RUN rm /etc/rhsm-host
```

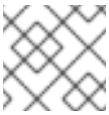
```
    RUN yum repolist --disablerepo=*
    RUN subscription-manager repos --enable rhocp-4.9-for-rhel-8-x86_64-rpms
    RUN yum -y update
    RUN yum install -y openshift-clients.x86_64
strategy:
  type: Docker
  dockerStrategy:
    volumes:
      - mounts:
          - destinationPath: "/etc/pki/entitlement"
        name: my-csi-shared-secret
        source:
          csi:
            driver: csi.sharedresource.openshift.io
            readOnly: true
            volumeAttributes:
              sharedSecret: my-share-bc
          type: CSI
```

4. **BuildConfig** オブジェクトからビルドを開始し、**oc** コマンドでログを追跡します。

oc start-build コマンドの例

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

例2.1 oc start-build コマンドからの出力例



注記

次の出力の一部のセクションは **...** に置き換えられました。

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

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

```
globally"
--> 1ef7c6d8c1a
STEP 3/9: RUN rm /etc/rhsm-host
time="2022-02-03T20:28:33Z" level=warning msg="Adding metacopy option, configured
globally"
--> b1c61f88b39
STEP 4/9: RUN yum repolist --disablerepo=*
Updating Subscription Management repositories.



...

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

...

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

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

...

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

Complete!
time="2022-02-03T20:29:19Z" level=warning msg="Adding metacopy option, configured
globally"
--> 609507b059e
STEP 8/9: ENV "OPENSHIFT_BUILD_NAME"="my-csi-bc-1"
"OPENSHIFT_BUILD_NAMESPACE"="my-csi-app-namespace"
--> cab2da3efc4
STEP 9/9: LABEL "io.openshift.build.name"="my-csi-bc-1"
"io.openshift.build.namespace"="my-csi-app-namespace"
```

```
COMMIT temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-1:edfe12ca
--> 821b582320b
Successfully tagged temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-
1:edfe12ca
821b582320b41f1d7bab4001395133f86fa9cc99cc0b2b64c5a53f2b6750db91
Build complete, no image push requested
```

2.10.6. 関連情報

- Insights Operator を使用した単純なコンテンツアクセス証明書のインポート
- 機能ゲートの使用による各種機能の有効化
- イメージストリームの管理
- ビルドストラテジー

2.11. ストラテジーによるビルドのセキュリティー保護

OpenShift Container Platform のビルドは特権付きコンテナーで実行されます。使用されるビルドストラテジーに応じて、権限がある場合は、ビルドを実行してクラスターおよびホストノードでの自らのパーミッションをエスカレートすることができます。セキュリティー対策として、ビルドを実行できるユーザーおよびそれらのビルドに使用されるストラテジーを制限します。カスタムビルドは特権付きコンテナー内で任意のコードを実行できるためにソースビルドより安全性が低くなります。そのためデフォルトで無効にされます。Dockerfile 処理ロジックにある脆弱性により、権限がホストノードで付与される可能性があるため、docker ビルドパーミッションを付与する際には注意してください。

デフォルトで、ビルドを作成できるすべてのユーザーには docker および Source-to-Image (S2I) ビルドストラテジーを使用するためにパーミッションが付与されます。クラスター管理者権限を持つユーザーは、ビルドストラテジーをユーザーにぐローバルに制限する方法についてのセクションで言及されているようにカスタムビルドストラテジーを有効にできます。

許可ポリシーを使用して、どのユーザーがどのビルドストラテジーを使用してビルドできるかについて制限することができます。各ビルドストラテジーには、対応するビルドサブリソースがあります。ストラテジーを使用してビルド作成するには、ユーザーにビルドを作成するパーミッションおよびビルドストラテジーのサブリソースで作成するパーミッションがなければなりません。ビルドストラテジーのサブリソースでの create パーミッションを付与するデフォルトロールが提供されます。

表2.3 ビルドストラテジーのサブリソースおよびロール

ストラテジー	サブリソース	ロール
Docker	ビルド/docker	system:build-strategy-docker
Source-to-Image (S2I)	ビルド/ソース	system:build-strategy-source
カスタム	ビルド/カスタム	system:build-strategy-custom
JenkinsPipeline	ビルド/jenkinspipeline	system:build-strategy-jenkinspipeline

2.11.1. ビルドストラテジーへのアクセスのグローバルな無効化

特定のビルドストラテジーへのアクセスをグローバルに禁止するには、クラスター管理者の権限を持つユーザーとしてログインし、**system:authenticated** グループから対応するロールを削除し、アノテーション **rbac.authorization.kubernetes.io/autoupdate: "false"** を適用してそれらを API の再起動間での変更から保護します。以下の例では、docker ビルドストラテジーを無効にする方法を示します。

手順

1. **rbac.authorization.kubernetes.io/autoupdate** アノテーションを適用します。

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

出力例

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

(1) **rbac.authorization.kubernetes.io/autoupdate** アノテーションの値を **"false"** に変更します。

2. ロールを削除します。

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

3. ビルドストラテジーのサブリソースもこれらのロールから削除されることを確認します。

```
$ oc edit clusterrole admin
```

```
$ oc edit clusterrole edit
```

4. ロールごとに、無効にするストラテジーのリソースに対応するサブリソースを指定します。

   a. **admin** の docker ビルドストラテジーの無効化

```
kind: ClusterRole
metadata:
  name: admin
...
- apiGroups:
  - ""
  - build.openshift.io
  resources:
  - buildconfigs
  - buildconfigs/webhooks
  - builds/custom 1
  - builds/source
  verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
...
```

1 **builds/custom** と **builds/source** を追加しして、**admin** ロールが割り当てられたユーザーに対して docker ビルドをグローバルに無効にします。

2.11.2. ユーザーへのビルドストラテジーのグルーバルな制限

一連の特定ユーザーのみが特定のストラテジーでビルドを作成できます。

前提条件

- ビルドストラテジーへのグローバルアクセスを無効にします。

手順

- ビルドストラテジーに対応するロールを特定ユーザーに割り当てます。たとえば、**system:build-strategy-docker** クラスターロールをユーザー **devuser** に追加するには、以下を実行します。

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

警告

ユーザーに対して **builds/docker** サブリソースへのクラスターレベルでのアクセスを付与することは、そのユーザーがビルドを作成できるすべてのプロジェクトにおいて、docker ストラテジーを使用してビルドを作成できることを意味します。

2.11.3. プロジェクト内でのユーザーへのビルドストラテジーの制限

ユーザーにビルドストラテジーをグローバルに付与するのと同様に、プロジェクト内の特定ユーザーのセットのみが特定ストラテジーでビルドを作成することを許可できます。

前提条件

- ビルドストラテジーへのグローバルアクセスを無効にします。

手順

- ビルドストラテジーに対応するロールをプロジェクト内の特定ユーザーに付与します。たとえば、プロジェクト **devproject** 内の **system:build-strategy-docker** ロールをユーザー **devuser** に追加するには、以下を実行します。

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

2.12. ビルド設定リソース

以下の手順でビルドを設定します。

2.12.1. ビルドコントローラー設定パラメーター

build.config.openshift.io/cluster リソースは以下の設定パラメーターを提供します。

パラメーター	説明
Build	ビルドの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。 **spec**: ビルドコントローラー設定のユーザーが設定できる値を保持します。
buildDefaults	ビルドのデフォルト情報を制御します。 **defaultProxy**: イメージのプルまたはプッシュ、およびソースのダウンロードを含む、ビルド操作のデフォルトのプロキシー設定が含まれます。 **BuildConfig** ストラテジーに **HTTP_PROXY**、**HTTPS_PROXY**、および **NO_PROXY** 環境変数を設定することで、値を上書きできます。 **gitProxy**: Git 操作のプロキシー設定のみが含まれます。設定されている場合、これは **git clone** などの Git コマンドのプロキシー設定を上書きします。 ここで設定されていない値は DefaultProxy から継承されます。 **env**: 指定される変数がビルドに存在しない場合にビルドに適用される一連のデフォルト環境変数。 **imageLabels**: 結果として生成されるイメージに適用されるラベルのリスト。**BuildConfig** に同じ名前のラベルを指定することでデフォルトのラベルを上書きできます。 **resources**: ビルドを実行するためのリソース要件を定義します。

パラメーター	説明
ImageLabel	**name**: ラベルの名前を定義します。ゼロ以外の長さを持つ必要があります。
buildOverrides	ビルドの上書き設定を制御します。 **imageLabels**: 結果として生成されるイメージに適用されるラベルのリスト。表にあるものと同じ名前のラベルを **BuildConfig** に指定する場合、ラベルは上書きされます。 **nodeSelector**: セレクター。 ビルド Pod がノードに適合させるには True である必要があります。 **tolerations**: ビルド Pod に設定された既存の容認を上書きする容認のリスト。
BuildList	**items**: 標準オブジェクトのメタデータ。

2.12.2. ビルド設定の設定

build.config.openshift.io/cluster リソースを編集してビルドの設定を行うことができます。

手順

- **build.config.openshift.io/cluster** リソースを編集します。

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

以下は、**build.config.openshift.io/cluster** リソースの例になります。

```
apiVersion: config.openshift.io/v1
kind: Build 1
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 2
  name: cluster
  resourceVersion: "107233"
  selfLink: /apis/config.openshift.io/v1/builds/cluster
  uid: e2e9cc14-78a9-11e9-b92b-06d6c7da38dc
spec:
  buildDefaults: 2
    defaultProxy: 3
      httpProxy: http://proxy.com
      httpsProxy: https://proxy.com
      noProxy: internal.com
    env: 4
    - name: envkey
      value: envvalue
    gitProxy: 5
      httpProxy: http://gitproxy.com
      httpsProxy: https://gitproxy.com
```

```
    noProxy: internalgit.com
  imageLabels:6
  - name: labelkey
    value: labelvalue
  resources:7
    limits:
      cpu: 100m
      memory: 50Mi
    requests:
      cpu: 10m
      memory: 10Mi
 buildOverrides:8
  imageLabels:9
  - name: labelkey
    value: labelvalue
  nodeSelector:10
    selectorkey: selectorvalue
  tolerations:11
  - effect: NoSchedule
    key: node-role.kubernetes.io/builds
operator: Exists
```

1. **Build**: ビルドの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。
2. **buildDefaults**: ビルドのデフォルト情報を制御します。
3. **defaultProxy**: イメージのプルまたはプッシュ、およびソースのダウンロードを含む、ビルド操作のデフォルトのプロキシー設定が含まれます。
4. **env**: 指定される変数がビルドに存在しない場合にビルドに適用される一連のデフォルト環境変数。
5. **gitProxy**: Git 操作のプロキシー設定のみが含まれます。設定されている場合、これは **git clone** などの Git コマンドのプロキシー設定を上書きします。
6. **imageLabels**: 結果として生成されるイメージに適用されるラベルのリスト。**BuildConfig** に同じ名前のラベルを指定することでデフォルトのラベルを上書きできます。
7. **resources**: ビルドを実行するためのリソース要件を定義します。
8. **buildOverrides**: ビルドの上書き設定を制御します。
9. **imageLabels**: 結果として生成されるイメージに適用されるラベルのリスト。表にあるものと同じ名前のラベルを **BuildConfig** に指定する場合、ラベルは上書きされます。
10. **nodeSelector**: セレクター。 ビルド Pod がノードに適合させるには True である必要があります。
11. **tolerations**: ビルド Pod に設定された既存の容認を上書きする容認のリスト。

2.13. ビルドのトラブルシューティング

ビルドの問題をトラブルシューティングするために、以下を使用します。

2.13.1. リソースへのアクセスのための拒否の解決

リソースへのアクセス要求が拒否される場合:

問題

ビルドが以下のエラーで失敗します。

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

解決策

プロジェクトに設定されているイメージのクォータのいずれかの上限を超えています。現在のクォータを確認して、適用されている制限数と、使用中のストレージを確認してください。

```
$ oc describe quota
```

2.13.2. サービス証明書の生成に失敗

リソースへのアクセス要求が拒否される場合:

問題

サービス証明書の生成は以下を出して失敗します (サービスの **service.beta.openshift.io/serving-cert-generation-error** アノテーションには以下が含まれます)。

出力例

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

解決策

証明書を生成したサービスがすでに存在しないか、サービスに異なる **serviceUID** があります。古いシークレットを削除し、サービスのアノテーション (**service.beta.openshift.io/serving-cert-generation-error** および **service.beta.openshift.io/serving-cert-generation-error-num**) をクリアして証明書の再生成を強制的に実行する必要があります。

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

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

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

注記

アノテーションを削除するコマンドでは、削除するアノテーション名の後に **-** を付けます。

2.14. ビルドの信頼される認証局の追加設定

以下のセクションを参照して、イメージレジストリーからイメージをプルする際に追加の認証局 (CA) がビルドによって信頼されるように設定します。

この手順を実行するには、クラスター管理者で **ConfigMap** を作成し、追加の CA を **ConfigMap** のキーとして追加する必要があります。

- **ConfigMap** は **openshift-config** namespace で作成される必要があります。
- **domain** は **ConfigMap** のキーであり、**value** は PEM エンコード証明書です。
  - それぞれの CA はドメインに関連付けられている必要があります。ドメインの形式は **hostname[..port]** です。
- **ConfigMap** 名は、**image.config.openshift.io/cluster** クラススコープ設定リソースの **spec.additionalTrustedCA** フィールドに設定される必要があります。

2.14.1. クラスターへの認証局の追加

以下の手順でイメージのプッシュおよびプル時に使用する認証局 (CA) をクラスターに追加することができます。

前提条件

- クラスター管理者の権限がある。
- レジストリーの公開証明書 (通常は、**/etc/docker/certs.d/** ディレクトリーにある **hostname/ca.crt** ファイル)。

手順

1. 自己署名証明書を使用するレジストリーの信頼される証明書が含まれる **ConfigMap** を **openshift-config** namespace に作成します。それぞれの CA ファイルで、**ConfigMap** のキーが **hostname[..port]** 形式のレジストリーのホスト名であることを確認します。

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

2. クラスターイメージの設定を更新します。

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

2.14.2. 関連情報

- **ConfigMap** の作成
- シークレットおよび **ConfigMap**
- カスタム PKI の設定

第3章 JENKINS から TEKTON への移行

3.1. JENKINS から TEKTON への移行

Jenkins と Tekton は、アプリケーションとプロジェクトのビルド、テスト、デプロイのプロセスを自動化するために使用されます。ただし、Tekton は、Kubernetes および OpenShift Container Platform とシームレスに動作するクラウドネイティブの CI/CD ソリューションです。本書は、Jenkins CI/CD ワークフローを Tekton に移行するのに役立ちます。

3.1.1. Jenkins と Tekton の概念の比較

本セクションでは、Jenkins と Tekton で使用される基本的な用語の概要を説明し、同等の用語を比較します。

3.1.1.1. Jenkins の用語

Jenkins は、共有ライブラリーおよびプラグインを使用して拡張可能な宣言型およびスクリプト化されたパイプラインを提供します。Jenkins における基本的な用語は以下のとおりです。

- **パイプライン**: Groovy 構文を使用してアプリケーションをビルドし、テストし、デプロイするプロセスをすべて自動化します。
- **ノード**: スクリプト化されたパイプラインのオーケストレーションまたは実行できるマシン。
- **ステージ**: パイプラインで実行されるタスクの概念的に異なるサブセット。プラグインまたはユーザーインターフェイスは、このブロックを使用してタスクの状態または進捗を表示します。
- **ステップ**: コマンドまたはスクリプトを使用して、実行する正確なアクションを指定する単一タスク。

3.1.1.2. Tekton の用語

Tekton は宣言型パイプラインに YAML 構文を使用し、タスクで設定されます。Tekton の基本的な用語は以下のとおりです。

- **パイプライン**: 一連のタスク、並行したタスク、またはその両方。
- **タスク**: コマンド、バイナリー、またはスクリプトとしてのステップシーケンス。
- **PipelineRun**: 1 つ以上のタスクを使用したパイプラインの実行。
- **TaskRun**: 1 つ以上のステップを使用したタスクの実行。

注記

パラメーターやワークスペースなどの入力のセットを使用して PipelineRun または TaskRun を開始し、実行結果を出力およびアーティファクトのセットで開始できます。

- **ワークスペース**: Tekton では、ワークスペースは以下の目的に対応する概念的なブロックです。
  - 入力、出力、およびビルドアーティファクトのストレージ。

- タスク間でデータを共有する一般的な領域。
- シークレットに保持される認証情報のマウントポイント、設定マップに保持される設定、および組織が共有される共通のツール。

注記

Jenkins には、Tekton ワークスペースに直接相当するものはありません。コントロールノードは、クローン作成したコードリポジトリー、ビルド履歴、およびアーティファクトを格納するため、ワークスペースと考えることができます。ジョブが別のノードに割り当てられると、クローンされたコードと生成されたアーティファクトがそのノードに保存されますが、ビルド履歴はコントロールノードによって維持されます。

3.1.1.3. 概念のマッピング

Jenkins と Tekton のビルディングブロックは同等ではなく、比較は技術的に正確なマッピングを提供しません。Jenkins と Tekton の次の用語と概念は、一般的に相関しています。

表3.1 Jenkins と Tekton: 基本的な比較

Jenkins	Tekton
パイプライン	パイプラインおよび PipelineRun
ステージ	タスク
Step	タスクのステップ

3.1.2. サンプルパイプラインの Jenkins から Tekton への移行

このセクションでは、Jenkins および Tekton でのパイプラインの例と同じ例を紹介します。これにより、ビルド、テスト、およびパイプラインを Jenkins から Tekton に移行するのに役立ちます。

3.1.2.1. Jenkins パイプライン

Groovy で書かれた Jenkins パイプラインについて、ビルド、テスト、およびデプロイについて見てみましょう。

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

```
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

3.1.2.2. Tekton パイプライン

Tekton では、Jenkins Pipeline の同等の例は 3 つのタスクで設定されており、それぞれは YAML 構文を使用して宣言的に記述できます。

build タスクの例

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

test タスクの例

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

deploy タスクの例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
```

```
  - name: source
  steps:
  - image: my-deploy-image
    command: ["make deploy"]
    workingDir: $(workspaces.source.path)
```

3 つのタスクを順次組み合わせ、Tekton パイプラインを形成できます。

例: ビルド、テスト、およびデプロイメント用の Tekton パイプライン

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

3.1.3. Jenkins プラグインから Tekton Hub タスクへの移行

プラグイン を使用して、Jenkins の機能を拡張することができます。Tekton で同様の拡張性を実現するには、Tekton Hub から利用可能なタスクのいずれかを使用します。

たとえば、Jenkins の git プラグイン に対応する Tekton Hub で利用可能な git-clone タスクについて考えてみましょう。

例: Tekton Hub からの git-clone タスク

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
 name: demo-pipeline
spec:
 params:
   - name: repo_url
   - name: revision
```

```
 workspaces:
   - name: source
 tasks:
   - name: fetch-from-git
     taskRef:
       name: git-clone
     params:
       - name: url
         value: $(params.repo_url)
       - name: revision
         value: $(params.revision)
     workspaces:
     - name: output
       workspace: source
```

3.1.4. カスタムタスクおよびスクリプトを使用した Tekton 機能の拡張

Tekton では、Tekton Hub で適切なタスクが見つからない場合、またはタスクをより細かく制御する必要がある場合は、カスタムタスクとスクリプトを作成して Tekton の機能を拡張できます。

例: maven test コマンドを実行するカスタムタスク

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

例: パスを指定してカスタムシェルスクリプトを実行します。

```
...
steps:
  image: ubuntu
  script: |
      #!/usr/bin/env bash
      /workspace/my-script.sh
...
```

例: YAML ファイルにカスタム Python スクリプトの実行

```
...
steps:
  image: python
  script: |
      #!/usr/bin/env python3
      print(“hello from python!”)
...
```

3.1.5. Jenkins および Tekton 実行モデルの比較

Jenkins と Tekton は同様の機能を提供しますが、アーキテクチャーと実行で異なります。このセクションでは、2 つの実行モデルを簡単に比較します。

表3.2 Jenkins および Tekton での実行モデルの比較

Jenkins	Tekton
Jenkins にはコントロールノードがあります。Jenkins は、パイプラインとステップを一元的に実行するか、他のノードで実行しているジョブのオーケストレーションを行います。	Tekton はサーバーレスおよび分散であり、実行のための中心的な依存関係はありません。
コンテナーは、パイプラインを使用してコントロールノードによって起動します。	Tekton では、container-first アプローチを採用しています。ここでは、すべてのステップが Pod で実行されるコンテナーとして実行されます (Jenkins のノードと同等)。
拡張性はプラグインを使用して実現します。	拡張性は、Tekton Hub のタスクを使用するか、カスタムタスクおよびスクリプトを作成して実行します。

3.1.6. 一般的な使用例の例

Jenkins と Tekton はどちらも、次のような一般的な CI/CD ユースケース向けの機能を提供します。

- Maven を使用したイメージのコンパイル、ビルド、およびデプロイ
- プラグインを使用してコア機能の拡張
- 共有可能なライブラリーおよびカスタムスクリプトの再利用

3.1.6.1. Jenkins と Tekton で Maven パイプラインの実行

Jenkins ワークフローと Tekton ワークフローの両方で Maven を使用して、イメージのコンパイル、ビルド、およびデプロイを行うことができます。既存の Jenkins ワークフローを Tekton にマッピングするには、次の例を検討してください。

例: Jenkins の maven を使用して、イメージをコンパイルおよびビルドし、OpenShift にデプロイします

```
#!/usr/bin/groovy
node('maven') {
   stage 'Checkout'
   checkout scm

   stage 'Build'
   sh 'cd helloworld && mvn clean'
   sh 'cd helloworld && mvn compile'

   stage 'Run Unit Tests'
   sh 'cd helloworld && mvn test'
```

```
    stage 'Package'
    sh 'cd helloworld && mvn package'

    stage 'Archive artifact'
    sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
    archive 'helloworld/target/*.war'

    stage 'Create Image'
    sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
    sh 'oc new-project helloworldproject'
    sh 'oc project helloworldproject'
    sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
    sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

    stage 'Deploy'
    sh 'oc new-app helloworld/jboss-eap70-deploy.json' }
```

例: イメージをコンパイルしてビルドし、Tekton の maven を使用して OpenShift にデプロイします。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: $(params.revision)
    - name: mvn-build
      taskRef:
        name: maven
      runAfter:
```

```
      - fetch-repo
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["-DskipTests", "clean", "compile"]
  - name: mvn-tests
    taskRef:
      name: maven
    runAfter:
      - mvn-build
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["test"]
  - name: mvn-package
    taskRef:
      name: maven
    runAfter:
      - mvn-tests
    workspaces:
      - name: source
        workspace: shared-workspace
      - name: maven-settings
        workspace: maven-settings
    params:
      - name: CONTEXT_DIR
        value: "$(params.context-path)"
      - name: GOALS
        value: ["package"]
  - name: create-image-and-deploy
    taskRef:
      name: openshift-client
    runAfter:
      - mvn-package
    workspaces:
      - name: manifest-dir
        workspace: shared-workspace
      - name: kubeconfig-dir
        workspace: kubeconfig-dir
    params:
      - name: SCRIPT
        value: |
          cd "$(params.context-path)"
          mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
```

```
oc new-project helloworldproject
oc project helloworldproject
oc process -f jboss-eap70-binary-build.json | oc create -f -
oc start-build eap-helloworld-app --from-dir=artifacts/
oc new-app jboss-eap70-deploy.json
```

3.1.6.2. プラグインを使用した Jenkins と Tekton のコア機能の拡張

Jenkins には、その広範なユーザーベースによって長年にわたって開発された多数のプラグインの大規模なエコシステムという利点があります。Jenkins プラグインインデックス でプラグインを検索および参照できます。

Tekton には、コミュニティーおよびエンタープライズユーザーによって開発および提供された多数のタスクもあります。再利用可能な Tekton タスクの公開されているカタログは、Tekton Hub で利用できます。

さらに、Tekton は、Jenkins エコシステムのプラグインの多くをコア機能に組み込んでいます。たとえば、承認は Jenkins と Tekton の両方で重要な機能です。Jenkins は ロールベースの Authorization Strategy プラグインを使用して認可を保証しますが、Tekton は OpenShift の組み込みロールベースアクセス制御システムを使用します。

3.1.6.3. Jenkins および Tekton での再利用可能なコードの共有

Jenkins 共有ライブラリー は、Jenkins パイプラインの一部に再利用可能なコードを提供します。ライブラリーは、Jenkinsfiles 間で共有され、コードの繰り返しなしに、高度にモジュール化されたパイプラインを作成します。

Tekton には Jenkins 共有ライブラリーの直接の機能は存在しませんが、カスタムタスクやスクリプトと組み合わせて Tekton Hub のタスクを使用して同様のワークフローを実行できます。

3.1.7. 関連情報

- ロールベースのアクセス制御

第4章 PIPELINES

4.1. 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 Pipelines の概要は、Understanding OpenShift Pipelines を参照してください。

4.1.1. 互換性およびサポート表

現在、今回のリリースに含まれる機能には テクノロジープレビュー のものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

以下の表では、機能は以下のステータスでマークされています。

TP	テクノロジープレビュー
GA	一般公開 (GA)

表4.1 互換性およびサポート表

Red Hat OpenShift Pipelines バージョン	コンポーネントのバージョン							OpenShift バージョン	サポートステータス
Operator	パイプライン	トリガー	CLI	カタログ	チェーン	ハブ	Pipelines as Code		
1.10	0.44.x	0.23.x	0.30.x	NA	0.15.x (TP)	1.12.x (TP)	0.17.x (GA)	4.10, 4.11, 4.12, 4.13	GA

Red Hat OpenShift Pipelines バージョン	コンポーネントのバージョン							OpenShift バージョン	サポートステータス
1.9	0.41.x	0.22.x	0.28.x	NA	0.13.x (TP)	1.11.x (TP)	0.15.x (GA)	4.10, 4.11, 4.12, 4.13	GA
1.8	0.37.x	0.20.x	0.24.x	NA	0.9.0 (TP)	1.8.x (TP)	0.10.x (TP)	4.10、4.11、4.12	GA
1.7	0.33.x	0.19.x	0.23.x	0.33	0.8.0 (TP)	1.7.0 (TP)	0.5.x (TP)	4.9、4.10、4.11	GA
1.6	0.28.x	0.16.x	0.21.x	0.28	該当なし	該当なし	該当なし	4.9	GA
1.5	0.24.x	0.14.x (TP)	0.19.x	0.24	該当なし	該当なし	該当なし	4.8	GA
1.4	0.22.x	0.12.x (TP)	0.17.x	0.22	該当なし	該当なし	該当なし	4.7	GA

さらに、ARM ハードウェアでの Red Hat OpenShift Pipeline の実行のサポートは、テクノロジープレビュー機能 としてご利用いただけます。

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。

4.1.2. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

4.1.3. Red Hat OpenShift Pipelines General Availability 1.10 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

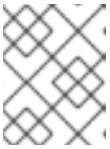
4.1.3.1. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.10 の主な新機能について説明します。

4.1.3.1.1. Pipelines

- 今回の更新により、**PipelineRun** または **TaskRun** Pod テンプレートで環境変数を指定して、タスクまたはステップで設定されている変数を上書きまたは追加できるようになりました。また、デフォルトの Pod テンプレートで環境変数を指定して、それらの変数をすべての **PipelineRuns** および **TaskRuns** に対してグローバルに使用することもできます。今回の更新では、Pod テンプレートからの伝播中に環境変数をフィルター処理する、**obhibited-envs** という名前の新しいデフォルト設定も追加されています。

- 今回の更新により、パイプラインのカスタムタスクがデフォルトで有効になります。



注記

この更新を無効にするには、**feature-flags** config カスタムリソースで **enable-custom-tasks** フラグを **false** に設定します。

- この更新プログラムは、カスタムタスクの **v1beta1.CustomRun** API バージョンをサポートします。

- 今回の更新により、カスタム実行を作成するための **PipelineRun** reconcilerのサポートが追加されました。たとえば、**custom-task-version** 機能フラグがデフォルト値の **v1alpha1** ではなく **v1beta1** に設定されている場合、**PipelineRuns** から作成されたカスタム **TaskRun** は**v1alpha1.Run** の代わりに **v1beta1.CustomRun** API バージョンを使用できるようになりました。

注記

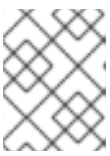
v1beta1.CustomRun 要求に応答するには 、***v1alpha1.Run** ではなく ***v1beta1.CustomRun** API バージョンをリッスンするようにカスタムタスクコントローラーを更新する必要があります。

- この更新により、新しい **retries** フィールドが **v1beta1.TaskRun** および **v1.TaskRun** 仕様に追加されます。

4.1.3.1.2. トリガー

- 今回の更新により、トリガーは、**v1beta1** API バージョンの **CustomRun** オブジェクトと共に、**v1** API バージョンの **Pipelines**、**Tasks**、**PipelineRuns**、および **TaskRuns** オブジェクトの作成をサポートします。

- 今回の更新により、GitHub Interceptor は、所有者または所有者による設定可能なコメントで呼び出されない限り、プルリクエストトリガーの実行をブロックします。



注記

この更新を有効または無効にするには、GitHub Interceptor 設定ファイルで **githubOwners** パラメーターの値を **true** または **false** に設定します。

- 今回の更新により、GitHub Interceptor は、プッシュおよびプルリクエストイベント用に変更されたすべてのファイルのコンマ区切りのリストを追加できるようになりました。変更されたファイルのリストは、最上位の拡張フィールドのイベントペイロードの **changed_files** がプロ

パティーに追加されます。

- 今回の更新により、TLS の **MinVersion** が **tls.VersionTLS12** に変更され、Federal Information Processing Standards (FIPS) モードが有効になっている場合に OpenShift Container Platform でトリガーが実行されるようになります。

4.1.3.1.3. CLI

- 今回の更新で、**Task**、**ClusterTask** または **Pipeline** が開始時に Container Storage Interface (CSI) ファイルをワークスペースとして渡すためのサポートが追加されました。
- この更新により、タスク、パイプライン、パイプライン実行、およびタスク実行リソースに関連付けられたすべての CLI コマンドに **v1** API サポートが追加されます。Tekton CLI は、これらのリソースの **v1beta1** と **v1** API の両方で動作します。
- 今回の更新で、**start** コマンドと **describe** コマンドにオブジェクトタイプパラメーターのサポートが追加されました。

4.1.3.1.4. Operator

- 今回の更新により、オプションのパイプラインプロパティーに **default-forbidden-env** パラメーターが追加されました。パラメーターには、Pod テンプレートを介して提供された場合に伝播されるべきではない、禁止された環境変数が含まれています。
- この更新により、Tekton Hub UI でのカスタムロゴのサポートが追加されます。カスタムロゴを追加するには、**customLogo** パラメーターの値を、Tekton Hub CR の base64 でエンコードされたロゴの URI に設定します。
- この更新により、git-clone タスクのバージョン番号が 0.9 に増加します。

4.1.3.1.5. Tekton Chains

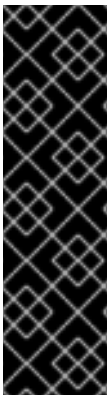
重要

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

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

- 今回の更新により、**PipelineRun** および **TaskRun** 設定証明にアノテーションとラベルが追加されました。
- この更新により、**slsa/v1** という名前の新しい形式が追加されます。これは、**in-toto** 形式で要求したときに生成されるものと同じ来歴を生成します。
- 今回の更新により、Sigstore 機能が実験的機能から除外されました。
- 今回の更新により、**predicate.materials** 関数に、**TaskRun** オブジェクトのすべてのステップとサイドカーからのイメージ URI とダイジェスト情報が含まれるようになりました。

4.1.3.1.6. Tekton Hub



重要

Tekton Hub はテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- この更新は、クラスターでの **v1** API バージョンの Tekton リソースのインストール、アップグレード、またはダウングレードをサポートします。
- この更新では、UI の Tekton Hub ロゴの代わりにカスタムロゴを追加できます。
- 今回の更新では、アーティファクトハブからリソースを取得してクラスターにインストールする **--type アーティファクト** フラグを追加することで、**tkn ハブインストール** コマンドの機能を拡張します。
- 今回の更新により、Artifact Hub からクラスターにインストールされるリソースにラベルとしてサポート層、カタログ、および組織情報が追加されます。

4.1.3.1.7. Pipelines as Code

- この更新により、着信 Webhook のサポートが強化されます。OpenShift Container Platform クラスターにインストールされた GitHub アプリケーションの場合、受信 Webhook に **git_provider** 仕様を提供する必要はありません。代わりに、Pipelines as Code がシークレットを検出し、それを着信 Webhook に使用します。
- 今回の更新により、同じトークンを使用して、GitHub 上の同じホストからデフォルト以外のブランチでリモートタスクを取得できるようになりました。
- 今回の更新により、Pipelines as Code は Tekton **v1** テンプレートをサポートします。**v1** および **v1beta1** テンプレートを使用できます。これは、Pipelines as Code が PR 生成のために読み取るものです。PR はクラスターで **v1** として作成されます。
- この更新の前は、OpenShift コンソール UI は、OpenShift namespace でランタイムテンプレートが見つからない場合、ハードコーディングされたパイプライン実行テンプレートをフォールバックテンプレートとして使用していました。**Pipelines-as-Code** config map のこの更新により、使用するコンソール用に、**pipelines-as-code-template-default** という名前の新しいデフォルトのパイプライン実行テンプレートが提供されます。
- 今回の更新により、Pipelines as Code は Tekton Pipelines 0.44.0 最小ステータスをサポートします。
- 今回の更新により、Pipelines as Code は Tekton **v1** API をサポートします。これは、Pipelines as Code が Tekton v0.44 以降と互換性を持つようになったことを意味します。
- 今回の更新により、OpenShift のコンソールと k8s の Tekton ダッシュボードの設定に加えて、カスタムコンソールダッシュボードを設定できるようになりました。
- 今回の更新により、Pipelines as Code は **tkn pac create repo** コマンドを使用して開始され

- 今回の更新により、Pipelines as Code は **tkn pac create repo** コマンドを使用して開始された GitHub アプリケーションのインストールを検出し、グローバルにインストールされている場合は GitHub Webhook を必要としません。

- この更新の前は、**PipelineRun** にアタッチされたタスクではなく **PipelineRun** の実行でエラーが発生した場合、Pipelines as Code は失敗を適切に報告しませんでした。今回の更新により、コードとしてのパイプラインは、**PipelineRun** を作成できなかった場合に GitHub チェックでエラーを適切に報告します。

- 今回の更新により、Pipelines as Code には、**PipelineRun** が実行される現在実行中の namespace にデプロイメントされる **target_namespace** 変数が含まれています。

- 今回の更新により、Pipelines as Code を使用すると、CLI ブートストラップ GitHub アプリケーションで GitHub エンタープライズの質問をバイパスできます。

- 今回の更新により、Pipelines as Code はリポジトリー CR が見つからない場合にエラーを報告しなくなりました。

- 今回の更新により、Pipelines as Code は、同じ名前の複数のパイプライン実行が見つかった場合にエラーを報告します。

4.1.3.2. 互換性を失わせる変更点

- 今回の更新により、以前のバージョンの **tkn** コマンドは Red Hat OpenShift Pipelines 1.10 と互換性がなくなりました。

- この更新により、Tekton CLI から **Cluster** および **CloudEvent** パイプラインリソースのサポートが削除されます。**tkn pipelineresource create** コマンドを使用してパイプラインリソースを作成することはできません。また、パイプラインリソースは、タスク、クラスタータスク、またはパイプラインの **start** コマンドでサポートされなくなりました。

- この更新により、Tekton Chains から来歴フォーマットとしての **tekton** が削除されます。

4.1.3.3. 非推奨および削除された機能

- Red Hat OpenShift Pipelines 1.10 では、**ClusterTask** コマンドが非推奨になり、将来のリリースで削除される予定です。**tkn task create** コマンドも、この更新で非推奨になりました。

- Red Hat OpenShift Pipelines 1.10 では、**v1** API がパイプラインリソースをサポートしていないため、**tkn task start** コマンドで使用されたフラグ **-i** および **-o** は非推奨になりました。

- Red Hat OpenShift Pipelines 1.10 では、**v1** API がパイプラインリソースをサポートしていないため、**tkn pipeline start** コマンドで使用されたフラグ **-r** は非推奨になりました。

- Red Hat OpenShift Pipelines 1.10 の更新では、**openshiftDefaultEmbeddedStatus** パラメーターが **full** 埋め込みステータスと **min** 埋め込みステータスの **both** に設定されます。デフォルトの埋め込みステータスを変更するフラグも非推奨であり、削除されます。さらに、パイプラインのデフォルトの埋め込みステータスは、将来のリリースで **minimal** に変更される予定です。

4.1.3.4. 既知の問題

- この更新には、以下の下位互換性のない変更が含まれています。

  - **PipelineResources** クラスターの削除

  - **PipelineResources** クラウドイベントの削除

- クラスターのアップグレード後にパイプラインメトリック機能が動作しない場合は、回避策として次のコマンドを実行します。

```
$ oc get tektoninstallersets.operator.tekton.dev | awk '/pipeline-main-static/ {print $1}' | xargs oc delete tektoninstallersets
```

- 今回の更新により、Crunchy PostgreSQL などの外部データベースの使用は、IBM Power、IBM Z、および {linuxoneProductName} ではサポートされなくなりました。代わりに、デフォルトの Tekton Hub データベースを使用してください。

4.1.3.5. 修正された問題

- この更新の前は、**opc pac** コマンドはヘルプを表示する代わりにランタイムエラーを生成していました。今回の更新により、**opc pac** コマンドがヘルプメッセージを表示するように修正されました。

- この更新の前は、**tkn pac create repo** コマンドを実行するには、リポジトリーを作成するための webhook の詳細が必要でした。今回の更新により、GitHub アプリケーションがインストールされている場合、**tkn-pac create repo** コマンドは Webhook を設定しません。

- この更新の前は、Tekton Pipelines で **PipelineRun** リソースの作成に問題があった場合、Pipelines as Code はパイプライン実行の作成エラーを報告しませんでした。たとえば、パイプラインの実行に存在しないタスクは、ステータスを表示しません。今回の更新により、Pipelines as Code は、欠落しているタスクとともに Tekton Pipelines からの適切なエラーメッセージを表示します。

- この更新プログラムは、認証が成功した後の UI ページのリダイレクトを修正します。これで、Tekton Hub にログインしようとしたのと同じページにリダイレクトされます。

- 今回の更新では、クラスタータスク、個々のタスク、およびパイプラインに対して、これらのフラグ **--all-namespaces** および **--output=yaml** を使用した **list** コマンドが修正されました。

- 今回の更新により、**repo.spec.url** URL の末尾にあるスラッシュが削除され、GitHub からの URL と一致するようになりました。

- この更新の前は、**marshalJSON** 関数はオブジェクトのリストをマーシャリングしませんでした。今回の更新で、**marshalJSON** 関数はオブジェクトのリストをマーシャリングします。

- 今回の更新により、Pipelines as Code を使用すると、CLI ブートストラップ GitHub アプリケーションで GitHub エンタープライズの質問をバイパスできます。

- この更新により、リポジトリーに 100 人を超えるユーザーがいる場合の GitHub コラボレーターチェックが修正されます。

- 今回の更新により、タスクまたはパイプラインの **sign** および **verify** コマンドは、kubernetes 設定ファイルなしで機能するようになりました。

- 今回の更新により、namespace でプルーナーがスキップされた場合、Tekton Operator は残りのプルーナー cron ジョブをクリーンアップします。

- この更新の前に、API **ConfigMap** オブジェクトは、カタログ更新間隔のユーザー設定値で更新されませんでした。この更新により、Tekon Hub CR の **CATALOG_REFRESH_INTERVAL** API が修正されます。

- この更新プログラムは、**EmbeddedStatus** 関数フラグを変更するときの **PipelineRunStatus** の調整を修正します。この更新により、次のパラメーターがリセットされます。

- **status.runs** および **status.taskruns** パラメーターを最小の **EmbeddedStatus** で **nil** に設定
- **full EmbeddedStatus** で **status.childReferences** パラメーターを **nil** に

- 今回の更新で、**ResolutionRequest** CRD に変換設定が追加されました。この更新により、**v1alpha1.ResolutionRequest** リクエストから **v1beta1.ResolutionRequest** リクエストへの変換が適切に設定されます。
- この更新プログラムは、パイプラインタスクに関連付けられている重複したワークスペースをチェックします。
- この更新により、コードでリゾルバーを有効にするためのデフォルト値が修正されます。
- この更新プログラムは、リゾルバーを使用した **TaskRef** および **PipelineRef** 名の変換を修正します。

4.1.3.6. Red Hat OpenShift Pipelines General Availability 1.10.1 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10.1 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.3.6.1. Pipelines as Code の修正された問題

- この更新の前は、ペイロードからのソースブランチ情報に **refs/heads/** が含まれていたが、ユーザーが設定したターゲットブランチにブランチ名 **main** のみが CEL 式に含まれていた場合、プッシュリクエストは失敗していました。今回の更新により、ベースブランチまたはターゲットブランチのペイロードに **refs/heads/** がある場合、Pipelines as Code はプッシュリクエストを渡し、パイプラインをトリガーします。
- この更新の前は、**PipelineRun** オブジェクトを作成できなかった場合、Tekton コントローラーから受け取ったエラーがユーザーに報告されませんでした。今回の更新により、Pipelines as Code はエラーメッセージを GitHub インターフェイスに報告し、ユーザーがエラーをトラブルシューティングできるようにします。Pipelines as Code は、パイプラインの実行中に発生したエラーも報告します。
- 今回の更新により、Pipelines as Code は、インフラストラクチャーの問題により OpenShift Container Platform クラスターでシークレットを作成できなかった場合に、シークレットを GitHub のチェックインターフェイスにエコーしません。
- 今回の更新により、使用されなくなった非推奨の API が Red Hat OpenShift Pipelines から削除されます。

4.1.3.7. Red Hat OpenShift Pipelines General Availability 1.10.2 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10.2 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.3.7.1. 修正された問題

この更新前は、Tekton Operator の問題により、ユーザーは **enable-api-fields** フラグの値を **beta** に設定できませんでした。今回の更新でこの問題が修正されています。**TektonConfig** CR で、**enable-api-fields** フラグの値を **beta** に設定できるようになりました。

4.1.3.8. Red Hat OpenShift Pipelines General Availability 1.10.3 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10.3 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.3.8.1. 修正された問題

この更新前は、Tekton Operator はカスタマイズのためのパフォーマンス設定フィールドを公開していませんでした。この更新により、クラスター管理者は、ニーズに基づいて **TektonConfig** CR の次のパフォーマンス設定フィールドをカスタマイズできます。

- **disable-ha**
- **buckets**
- **kube-api-qps**
- **kube-api-burst**
- **threads-per-controller**

4.1.3.9. Red Hat OpenShift Pipelines General Availability 1.10.4 のリリースノート

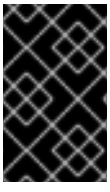
今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10.4 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.3.9.1. 修正された問題

- この更新により、パイプライン実行における **PipelineRef** フィールドのバンドルリゾルバー変換の問題が修正されます。現在、変換機能は、変換後に **kind** フィールドの値を **Pipeline** に設定します。
- この更新前は、**pipelinerun.timeouts** フィールドは **timeouts.pipeline** 値にリセットされ、**timeouts.tasks** 値と **timeouts.finally** 値は無視されました。この更新により問題が修正され、**PipelineRun** リソースの正しいデフォルトのタイムアウト値が設定されます。
- この更新前は、コントローラーのログに不要なデータが含まれていました。今回の更新でこの問題が修正されています。

4.1.3.10. Red Hat OpenShift Pipelines General Availability 1.10.5 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.10.5 が OpenShift Container Platform 4.11、4.12、4.13 に加え、4.10 でも利用できるようになりました。



重要

Red Hat OpenShift Pipelines 1.10.5 は、OpenShift Container Platform 4.10、4.11、4.12、および 4.13 の **pipelines-1.10** チャネルでのみ使用できます。OpenShift Container Platform バージョンの **latest** チャネルでは利用できません。

4.1.3.10.1. 修正された問題

- この更新が行われる前は、**oc** および **tkn** コマンドを使用しても、大規模なパイプライン実行がリストされたり、削除されませんでした。この更新では、この問題の原因となっていた巨大なアノテーションを圧縮することで、この問題を軽減します。圧縮後もパイプラインの実行が大きすぎる場合は、同じエラーが再発することに注意してください。

- この更新より前は、**pipelineRun.spec.taskRunSpecs.podTemplate** オブジェクトで指定された Pod テンプレートのみがパイプライン実行の対象となります。この更新により、**pipelineRun.spec.podTemplate** オブジェクトで指定された Pod テンプレートも考慮され、**pipelineRun.spec.taskRunSpecs.podTemplate** オブジェクトで指定されたテンプレートとマージされます。

4.1.4. Red Hat OpenShift Pipelines General Availability 1.9 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.9 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.4.1. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.9 の主な新機能について説明します。

4.1.4.1.1. Pipelines

- 今回の更新により、パイプラインパラメーターと結果を配列とオブジェクトディクショナリー形式で指定できるようになりました。
- この更新により、Container Storage Interface (CSI) およびワークスペースの projected ボリュームがサポートされます。
- 今回の更新により、パイプラインステップを定義するときに **stdoutConfig** および **stderrConfig** パラメーターを指定できるようになりました。これらのパラメーターを定義すると、ステップに関連付けられた標準出力と標準エラーをローカルファイルにキャプチャーするのに役立ちます。
- 今回の更新により、**steps.onError** イベントハンドラーに **$(params.CONTINUE)** などの変数を追加できるようになりました。
- 今回の更新により、**PipelineResults** 定義で **finally** タスクからの出力を使用できるようになりました。たとえば **$(finally.<pipelinetask-name>.result.<result-name>)** では、**<pipelinetask-name>** はパイプラインタスク名を表し、**<result-name>** は結果名を表します。
- この更新では、タスク実行のタスクレベルのリソース要件をサポートがされます。
- 今回の更新により、名前に基づいて、パイプラインと定義されたタスクの間で共有されるパラメーターを再作成する必要がなくなりました。この更新は、開発者プレビュー機能の一部です。
- この更新により、組み込みの git、クラスター、バンドル、およびハブリゾルバーなどのリモート解決のサポートが追加されます。

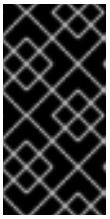
4.1.4.1.2. トリガー

- 今回の更新では、**NamespacedInterceptor** を定義する **Interceptor** CRD が追加されました。**NamespacedInterceptor** は、トリガー内のインターセプター参照の **kind** セクションまたは **EventListener** 仕様で使用できます。
- この更新により **CloudEvents** が有効になります。
- 今回の更新により、トリガーを定義するときに Webhook ポート番号を設定できるようになりました。

- 今回の更新では、トリガー **eventID** を使用した **TriggerBinding** への入力がサポートされるようになりました。
- この更新では、**ClusterInterceptor** サーバーの証明書の検証とローテーションがサポートされています。
    - トリガーは、コアインターセプターの証明書を検証し、証明書の有効期限が切れると新しい証明書を **ClusterInterceptor** にローテーションします。

4.1.4.1.3. CLI

- 今回の更新では、**describe** コマンドでのアノテーションの表示がサポートされています。
- 今回の更新では、**pr describe** コマンドでのパイプライン、タスク、およびタイムアウトの表示がサポートされています。
- 今回の更新では、**pipeline start** コマンドでパイプライン、タスク、およびタイムアウトを提供するフラグが追加されました。
- 今回の更新では、タスクとパイプラインの **describe** コマンドで、オプションまたは必須のワークスペースの存在を表示できるようになりました。
- 今回の更新では、タイムスタンプ付きのログを表示するための **timestamps** フラグが追加されました。
- 今回の更新では、**PipelineRun** に関連付けられた **TaskRun** の削除を無視する新しいフラグ **--ignore-running-pipelinerun** が追加されました。
- 今回の更新では、実験的なコマンドのサポートが追加されました。今回の更新では、試験的なサブコマンドである **sign** と **verify** も **tkn** CLI ツールに追加されました。
- 今回の更新では、ファイルを生成せずに Z シェル (Zsh) 補完機能を使用できるようになりました。
- 今回の更新では、**opc** という新しい CLI ツールが導入されました。今後のリリースで、**tkn** CLI ツールが **opc** に置き換えられることが予想されます。



重要

- 新しい CLI ツール **opc** はテクノロジープレビュー機能です。
- **opc** は **tkn** の代替となり、Red Hat OpenShift Pipelines 固有の追加機能を備えていますが、それらは必ずしも **tkn** に適合するとは限りません。

4.1.4.1.4. Operator

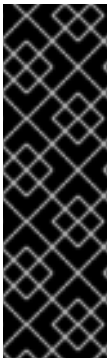
- 今回の更新により、Pipelines as Code がデフォルトでインストールされます。**-p** フラグを使用して、Pipelines as Code を無効にすることができます。

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": false}}}}}'
```

- 今回の更新により、**TektonConfig** CRD で Pipelines as Code 設定の変更も可能になりました。
- 今回の更新により、開発者パースペクティブを無効にした場合に Operator が開発者コンソール関連のカスタムリソースをインストールしなくなりました。

- 今回の更新には、Bitbucket Server および Bitbucket Cloud の **ClusterTriggerBinding** サポートが含まれており、クラスター全体で **TriggerBinding** を再利用するのに役立ちます。

4.1.4.1.5. リゾルバー



重要

リゾルバーはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

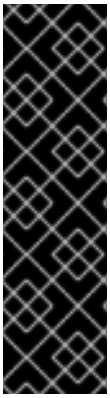
- 今回の更新により、**TektonConfig** CRD でパイプラインリゾルバーを設定できるようになりました。パイプラインリゾルバー **enable-bundles-resolver**、**enable-cluster-resolver**、**enable-git-resolver**、**enable-hub-resolver** を、有効または無効にできます。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    enable-bundles-resolver: true
    enable-cluster-resolver: true
    enable-git-resolver: true
    enable-hub-resolver: true
...
```

TektonConfig でリゾルバー固有の設定も指定できます。たとえば、次のフィールドを **mapstringstring** 形式で定義して、個々のリゾルバーを設定できます。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines
    cluster-resolver-config:
      default-namespace: test
    git-resolver-config:
      server-url: localhost.com
    hub-resolver-config:
      default-tekton-hub-catalog: tekton
...
```

4.1.4.1.6. Tekton Chains



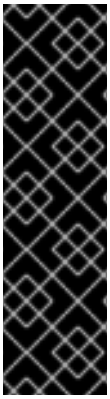
重要

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

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

- この更新の前は、Open Container Initiative (OCI) イメージのみが in-toto 出所エージェントの **TaskRun** の出力としてサポートされていました。この更新では、**ARTIFACT_URI** および **ARTIFACT_DIGEST** の接尾辞を使用して、出所メタデータが出力として追加されます。
- この更新の前は、**TaskRun** 構成証明のみがサポートされていました。この更新では、**PipelineRun** 構成証明のサポートも追加されます。
- この更新では、Pod テンプレートから **imgPullSecret** パラメーターを取得するための Tekton Chains のサポートが追加されます。この更新により、サービスアカウントを変更せずに、各パイプライン実行またはタスク実行に基づいてリポジトリー認証を設定できます。

4.1.4.1.7. Tekton Hub



重要

Tekton Hub はテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- この更新では、管理者は、デフォルトの Tekton Hub データベースを使用する代わりに、Crunchy PostgreSQL などの外部データベースを Tekton Hub で 使用できるようになりました。この更新は、次のアクションを実行するのに役立ちます。
  - Tekton Hub で使用する外部データベースの座標指定。
  - Operator によってデプロイされたデフォルトの Tekton Hub データベースの無効化。
- この更新では、外部 Git リポジトリーから **config.yaml** の依存関係が削除され、完全な設定データが API **ConfigMap** に移動されます。この更新は、管理者が次のアクションを実行するのに役立ちます。
  - Tekton Hub カスタムリソースへの、カテゴリー、カタログ、スコープ、defaultScopes などの設定データの追加。
  - クラスター上の Tekton Hub 設定データの変更。すべての変更は、Operator をアップグレードしても保持されます。
  - Tekton Hub のカタログリストの更新。

- Tekton Hub のカテゴリーの変更。

注記

設定データを追加しない場合は、Tekton Hub 設定用の API **ConfigMap** のデフォルトデータを使用できます。

4.1.4.1.8. Pipelines as Code

- この更新では、**Repository** CRD で同時実行制限のサポートが追加され、一度にリポジトリーで実行される **PipelineRuns** の最大数が定義できます。プルリクエストまたはプッシュイベントからの **PipelineRun** は、アルファベット順にキューに入れられます。
- この更新では、リポジトリーの最新パイプライン実行のログを表示するための新しいコマンド **tkn pac logs** が追加されます。
- この更新では、GitHub および GitLab へのプッシュリクエストとプルリクエストのファイルパスにおける高度なイベントマッチングがサポートされています。たとえば、**docs** ディレクトリー内のマークダウンファイルのパスが変更された場合にのみ、Common Expression Language (CEL) を使用してパイプラインを実行できます。

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
   event == "pull_request" && "docs/*.md".pathChanged()
```

- 今回の更新により、アノテーションを使用して、**pipelineRef:** オブジェクトでリモートパイプラインを参照できるようになります。
- 今回の更新により、Pipelines as Code を使用して新しい GitHub リポジトリーを自動設定できるようになります。これにより、namespace が設定され、GitHub リポジトリーの **Repository** CRD が作成されます。
- 今回の更新により、Pipelines as Code は、プロバイダー情報を使用して **PipelineRuns** のメトリクスを生成します。
- この更新では、**tkn-pac** プラグインに次の機能拡張が提供されます。
  - 実行中のパイプラインを正しく検出します。
  - 障害完了時間がない場合に期間表示を修正します。
  - エラースニペットを表示し、**tkn-pac describe** コマンドのエラー正規表現パターンを強調表示します。
  - **use-real-time** スイッチを **tkn-pac ls** および **tkn-pac describe** コマンドに追加します。
  - **tkn-pac** ログのドキュメントをインポートします。
  - **tkn-pac ls** および **tkn-pac describe** コマンドで、**pipelineruntimeout** を失敗として表示します。
  - **--target-pipelinerun** オプションを使用して、特定のパイプライン実行の失敗を表示します。

今回の更新により、バージョン管理システム (VCS) イベントをは Git...

- 今回の更新により、バージョン管理システム (VCS) コメントまたは GitHub チェックの小さなスニペットの形式で、パイプライン実行のエラーを表示できます。
- 今回の更新により、Pipelines as Code は、タスクが単純な形式である場合にオプションでタスク内のエラーを検出し、それらのタスクを GitHub のアノテーションとして追加できます。この更新は、開発者プレビュー機能の一部です。
- この更新では、次の新しいコマンドが追加されます。
  - **tkn-pac webhook add**: プロジェクトリポジトリー設定に Webhook を追加し、リポジトリーを更新せずに、既存の **k8s Secret** オブジェクトの **webhook.secret** キーを更新します。
  - **tkn-pac webhook update-token**: リポジトリーを更新せずに、既存の **k8s Secret** オブジェクトのプロバイダートークンを更新します。
- この更新により、**tkn-pac create repo** コマンドの機能が強化されます。このコマンドは、GitHub、GitLab、および BitbucketCloud の Webhook を作成および設定し、リポジトリーを作成します。
- この更新により、**tkn-pac describe** コマンドは 50 件の最新イベントが順に表示されます。
- この更新では、**tkn-pac logs** コマンドに **--last** オプションが追加されます。
- この更新により、**tkn-pac resolve** コマンドは、ファイルテンプレートで **git_auth_secret** を検出すると、トークンの入力を求めます。
- この更新により、Pipelines as Code はシークレットをログスニペットから非表示にして、GitHub インターフェイスでシークレットが公開されるのを回避します。
- この更新により、**git_auth_secret** に対して自動的に生成されるシークレットは、**PipelineRun** による所有者参照になります。シークレットは、パイプライン実行の実行後ではなく、**PipelineRun** で消去されます。
- この更新により、**/cancel** コメントを使用したパイプライン実行のキャンセルがサポートされます。
- この更新の前は、GitHub アプリのトークンスコープが定義されておらず、すべてのリポジトリーインストールでトークンが使用されていました。この更新により、次のパラメーターを使用して、GitHub アプリトークンの範囲をターゲットリポジトリーに設定できます。
  - **secret-github-app-token-scoped**: アプリのインストールがアクセスできるすべてのリポジトリーではなく、ターゲットリポジトリーにアプリトークンのスコープを設定します。
  - **secret-github-app-scope-extra-repos**: 追加の所有者またはリポジトリーを使用して、アプリトークンのスコープをカスタマイズします。
- この更新により、GitLab でホストされている独自の Git リポジトリーで Pipelines as Code を使用できるようになります。
- この更新により、namespace の kubernetes イベント形式でパイプライン実行の詳細にアクセスできるようになります。その詳細は、admin namespace へのアクセスを必要とせずにパイプラインエラーをトラブルシューティングするのに役立ちます。
- この更新により、Git プロバイダーを使用した Pipelines as Code での URL 認証がサポートされます。

この更新により、**pipelines-as-code-configmap** の設定を使用して、ビルドタスクログの名前を設

- この更新により、**pipelines-as-code** config map の設定を使用して、ハブカタログの名前を設定できるようになります。
- この更新により、**max-keep-run** パラメーターの上限とデフォルトの制限を設定できるようになります。
- 今回の更新では、Pipelines as Code にカスタム Secure Sockets Layer (SSL) 証明書を挿入し、カスタム証明書を使用してプロバイダーインスタンスに接続する方法を説明したドキュメントが追加されます。
- この更新により、**PipelineRun** リソース定義にログ URL がアノテーションとして含まれるようになります。たとえば、**tkn-pac describe** コマンドは、**PipelineRun** を記述するときにログリンクを表示します。
- 今回の更新により、**tkn-pac** ログに **PipelineRun** 名ではなくリポジトリー名が表示されるようになります。

4.1.4.2. 互換性を失わせる変更点

- 今回の更新では、**Conditions** カスタムリソース定義 (CRD) タイプが削除されました。代わりに **WhenExpressions** を使用します。
- 今回の更新では、Pipeline、PipelineRun、Task、Clustertask、TaskRun などの **tekton.dev/v1alpha1** API パイプラインリソースのサポートが削除されました。
- 今回の更新では、**tkn-pac setup** コマンドが削除されました。代わりに、**tkn-pac webhook add** コマンドを使用して、Webhook を既存の Git リポジトリーに再度追加します。また、**tkn-pac webhook update-token** コマンドを使用して、Git リポジトリー内の既存のシークレットオブジェクトの個人プロバイダーアクセストークンを更新します。
- 今回の更新により、デフォルト設定でパイプラインを実行する namespace は、**pod-security.kubernetes.io/enforce:privileged** ラベルをワークロードに適用しません。

4.1.4.3. 非推奨および削除された機能

- Red Hat OpenShift Pipelines 1.9.0 リリースでは、**ClusterTasks** が非推奨となり、今後のリリースで削除される予定です。代わりに、**Cluster Resolver** を使用できます。
- Red Hat OpenShift Pipelines 1.9.0 リリースでは、単一の **EventListener** 仕様で **triggers** と **namespaceSelector** フィールドを使用することは推奨されておらず、今後のリリースで削除される予定です。これらのフィールドは、異なる **EventListener** 仕様では正常に使用できます。
- Red Hat OpenShift Pipelines 1.9.0 リリースでは、**tkn pipelinerun describe** コマンドは **PipelineRun** リソースのタイムアウトを表示しません。
- Red Hat OpenShift Pipelines 1.9.0 リリースでは、PipelineResource カスタムリソース (CR) が非推奨になりました。**PipelineResource** CR はテクノロジープレビュー機能であり、**tekton.dev/v1alpha1** API の一部でした。
- Red Hat OpenShift Pipelines 1.9.0 リリースでは、クラスタータスクからのカスタムイメージパラメーターは非推奨になりました。代わりとして、クラスタータスクをコピーして、その中でカスタムイメージを使用できます。

4.1.4.4. 既知の問題

- Red Hat OpenShift Pipelines Operator をアンインストールすると、**chains-secret** および **chains-config** config map が削除されます。これらにはユーザーデータが含まれているため、削除せずに保持する必要があります。

- Windows でコマンドの **tkn pac** セットを実行すると、**Command finished with error: not supported by Windows.** のエラーメッセージが表示される場合があります。
回避策: **NO_COLOR** 環境変数を **true** に設定します。

- **tkn pac resolve** コマンドがテンプレート化されたパラメーター値を使用して機能する場合、**tkn pac resolve -f <filename> | oc create -f** コマンドを実行しても、想定どおりの結果が得られない場合があります。
回避策: この問題を軽減するには、**tkn pac resolve -f <filename> -o tempfile.yaml** コマンドを実行して **tkn pac resolve** の出力を一時ファイルに保存してから、**oc create -f tempfile.yaml** コマンドを実行します。例: **tkn pac resolve -f <filename> -o /tmp/pull-request-resolved.yaml && oc create -f /tmp/pull-request-resolved.yaml**。

4.1.4.5. 修正された問題

- この更新の前は、空の配列を置き換えた後、元の配列は中のパラメーターを無効にして空の文字列を返していました。今回の更新により、この問題は解決され、元の配列は空として返されます。

- この更新の前は、パイプライン実行のサービスアカウントに重複するシークレットが存在すると、タスク Pod の作成に失敗していました。今回の更新により、この問題が解決され、サービスアカウントに重複するシークレットが存在する場合でもタスク Pod は正常に作成されるようになりました。

- この更新の前は、TaskRun の **spec.StatusMessage** フィールドを見ても、**TaskRun** がユーザーによってキャンセルされたのか、その一部である **PipelineRun** によってキャンセルされたのかを区別できませんでした。今回の更新により、この問題は解決され、ユーザーは TaskRun の **spec.StatusMessage** フィールドを見て、**TaskRun** のステータスを区別できるようになりました。

- この更新の前は、無効なオブジェクトの古いバージョンを削除すると、webhook の検証が削除されていました。今回の更新で、この問題は解決されました。

- 今回の更新の前は、**timeouts.pipeline** パラメーターを **0** に設定すると、**timeouts.tasks** パラメーターまたは **timeouts.finally** パラメーターを設定できませんでした。今回の更新で問題が解決されました。これで、**timeouts.pipeline** パラメーター値を設定するときに、`timeouts.tasks` パラメーターまたは **timeouts.finally** パラメーターのいずれかの値を設定できます。以下に例を示します。

```
yaml
kind: PipelineRun
spec:
  timeouts:
    pipeline: "0"  # No timeout
    tasks: "0h3m0s"
```

- この更新の前は、別のツールが PipelineRun または TaskRun のラベルまたはアノテーションを更新すると、競合状態が発生する可能性がありました。今回の更新により、この問題は解決され、ラベルまたはアノテーションを結合できるようになりました。

- この更新の前は、ログキーにパイプラインコントローラーと同じキーはありませんでした。今回の更新により、この問題は解決され、パイプラインコントローラーのログストリームと一致するようにログキーが更新されました。ログのキーは、ts から timestamp、level から

severity、message から msg に変更されました。

- この更新の前は、PipelineRun が不明ステータスで削除された場合、エラーメッセージは生成されませんでした。今回の更新により、この問題は解決され、エラーメッセージが生成されるようになります。

- この更新の前は、**list** や **push** などのバンドルコマンドにアクセスするには、**kubeconfig** ファイルを使用する必要がありました。今回の更新により、この問題は解決され、**kubeconfig** ファイルはバンドルコマンドにアクセスする必要がなくなりました。

- この更新の前は、TaskRun の削除中に親の PipelineRun が実行されていた場合、TaskRun が削除されていました。今回の更新により、この問題は解決され、親 PipelineRun が実行されていても TaskRuns は削除されなくなりました。

- この更新の前は、ユーザーがパイプラインコントローラーで許可されているよりも多くのオブジェクトを含むバンドルのビルドを試みた場合、Tekton CLI はエラーメッセージを表示しませんでした。今回の更新により、この問題は解決され、ユーザーがパイプラインコントローラーで許可されている制限を超える数のオブジェクトを含むバンドルを構築しようとすると、Tekton CLI にエラーメッセージが表示されるようになります。

- この更新の前は、クラスターから namespace が削除されても、operator は **ClusterInterceptor ClusterRoleBinding** サブジェクトから namespace を削除しませんでした。今回の更新により、この問題は解決され、operator は **ClusterInterceptor ClusterRoleBinding** サブジェクトから namespace を削除するようになります。

- この更新の前は、デフォルトの Red Hat OpenShift Pipelines Operator インストールで、**pipelines-scc-rolebinding security context constraint** (SCC) ロールバインディングリソースがクラスターに残りました。今回の更新により、デフォルトの Red Hat OpenShift Pipelines Operator インストールで、**pipelines-scc-rolebinding security context constraint** (SCC) ロールバインディングリソースがクラスターから削除されるようになります。

- この更新の前は、Pipelines as Code は Pipelines as Code **ConfigMap** オブジェクトから更新された値を取得しませんでした。今回の更新により、この問題は修正され、Pipelines as Code **ConfigMap** オブジェクトが新しい変更を検索するようになります。

- この更新の前は、Pipelines as Code コントローラーは **tekton.dev/pipeline** ラベルが更新されるのを待たずに **checkrun id** ラベルを追加して、競合状態を引き起こしていました。今回の更新により、Pipelines as Code コントローラーは **tekton.dev/pipeline** ラベルが更新されるのを待ってから **checkrun id** ラベルを追加するようになりました。これは、競合状態の回避に役立ちます。

- この更新の前は、git リポジトリーに **PipelineRun** がすでに存在する場合、**tkn-pac create repo** コマンドはそれをオーバーライドしませんでした。今回の更新では **tkn-pac create** コマンドが修正され、git リポジトリーに **PipelineRun** が存在する場合はそれをオーバーライドするようになり、問題は解決されました。

- この更新の前は、**tkn pac describe** コマンドはすべてのメッセージの理由を表示しませんでした。今回の更新により、この問題は修正され、**tkn pac describe** コマンドはすべてのメッセージの理由を表示するようになります。

- この更新の前は、アノテーションのユーザーが **refs/head/rel-*** などの正規表現形式を使用して値を指定した場合、プルリクエストは失敗していました。ベースブランチに **refs/heads** がないため、プルリクエストは失敗していました。今回の更新では接頭辞が追加され、一致するかどうかもチェックされます。これで問題が解決し、プルリクエストが成功するようになります。

4.1.4.6. Red Hat OpenShift Pipelines General Availability 1.9.1 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.9.1 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.4.7. 修正された問題

- この更新の前は、**tkn pac repo list** コマンドは Microsoft Windows で実行できませんでした。今回の更新で問題が修正され、Microsoft Windows で **tkn pac repo list** コマンドを実行できるようになりました。

- この更新の前は、Pipelines as Code ウォッチャーは設定変更イベントをすべて受信するわけではありませんでした。今回の更新により、Pipelines as Code ウォッチャーが更新され、Pipelines as Code ウォッチャーが設定変更イベントを見逃さなくなりました。

- この更新の前は、Pipelines as Code によって作成された **TaskRuns** や **PipelineRuns** などの Pod は、クラスター内のユーザーによって公開されたカスタム証明書にアクセスできませんでした。今回の更新で問題が修正され、クラスター内で **TaskRuns** または **PipelineRuns** Pod からカスタム証明書にアクセスできるようになりました。

- この更新の前は、FIPS が有効になっているクラスターで、**Trigger** リソースで使用される **tekton-triggers-core-interceptors** コアインターセプターは、Pipelines Operator がバージョン 1.9 にアップグレードされた後に機能しませんでした。今回の更新で問題が解決されました。現在、OpenShift はすべてのコンポーネントに MInTLS 1.2 を使用しています。その結果、**tekton-triggers-core-interceptors** コアインターセプターが TLS バージョン 1.2 に更新され、その機能は正確に実行されるようになりました。

- この更新の前は、内部 OpenShift イメージレジストリーでパイプライン実行を使用する場合、パイプライン実行定義でイメージへの URL をハードコーディングする必要がありました。以下に例を示します。

  ```
  ...
    - name: IMAGE_NAME
      value: 'image-registry.openshift-image-
  registry.svc:5000/<test_namespace>/<test_pipelinerun>'
  ...
  ```

  Pipelines as Code のコンテキストでパイプライン実行を使用する場合、ハードコーディングされた値により、異なるクラスターおよび namespace でパイプライン実行定義をしようできませんでした。

  今回の更新により、namespace とパイプライン実行名の値をハードコーディングする代わりに動的テンプレート変数を使用して、パイプライン実行定義を一般化できます。以下に例を示します。

  ```
  ...
    - name: IMAGE_NAME
      value: 'image-registry.openshift-image-registry.svc:5000/{{ target_namespace
  }}/$(context.pipelineRun.name)'
  ...
  ```

- この更新の前は、Pipelines as Code は同じ GitHub トークンを使用して、デフォルトの GitHub ブランチの同じホストでのみ使用可能なリモートタスクを取得していました。今回の更新で問題が解決されました。Pipelines as Code は同じ GitHub トークンを使用して、任意の GitHub ブランチからリモートタスクを取得するようになりました。

4.1.4.8. 既知の問題

- Tekton Hub CR で使用される Hub API **ConfigMap** オブジェクト内のフィールドである **CATALOG_REFRESH_INTERVAL** の値が、ユーザーが指定したカスタム値で更新されません。
  回避策: なし。問題 SRVKP-2854 を確認してください。

4.1.4.9. 互換性を失わせる変更点

- 今回の更新で、OpenShift Container Platform のアップグレードを妨げる OLM のご設定の問題が発生しました。この問題は今後のリリースで修正される予定です。

4.1.4.10. Red Hat OpenShift Pipelines General Availability 1.9.2 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.9.2 が OpenShift Container Platform 4.11、4.12、および 4.13 で利用できるようになりました。

4.1.4.11. 修正された問題

- この更新前は、リリースの以前のバージョンで OLM の誤設定の問題が発生しており、OpenShift Container Platform のアップグレードが妨げられていました。今回の更新により、この誤設定の問題が修正されました。

4.1.4.12. Red Hat OpenShift Pipelines General Availability 1.9.3 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.9.3 が OpenShift Container Platform 4.11、4.12、4.13 に加え、4.10 でも利用できるようになりました。

4.1.4.13. 修正された問題

- 今回の更新により、大規模パイプラインのパフォーマンスの問題が修正されました。これにより、CPU 使用率は 61%、メモリー使用率は 44% 削減されました。

- この更新前は、**when** 式が原因でタスクが実行されない場合、パイプラインの実行は失敗していました。今回の更新により、パイプライン結果でスキップされたタスクの結果が検証されないようにすることで問題を修正しました。現在は、パイプラインの結果は出力されず、結果の欠落を原因とするパイプライン実行の失敗は発生しません。

- 今回の更新により、**pipelineref.bundle** を **v1beta1** API のバンドルリゾルバーに変換する動作が修正されました。現在は変換機能により、変換後に **kind** フィールドの値が **Pipeline** に設定されます。

- この更新前は、Pipelines Operator の問題により、ユーザーは **spec.pipeline.enable-api-fields** フィールドの値を **beta** に設定できませんでした。今回の更新でこの問題が修正されています。現在は、**TektonConfig** カスタムリソースで値を **alpha**、**stable**、**beta** に設定できます。

- この更新前は、Pipelines as Code はクラスターエラーが原因でシークレットを作成できなかった場合、GitHub チェック実行でパブリックな一時トークンが表示されていました。今回の更新でこの問題が修正されています。現在は、シークレットの作成に失敗しても、GitHub チェックインターフェイスにトークンは表示されません。

4.1.4.14. 既知の問題

- 現在、OpenShift Container Platform Web コンソールでのパイプライン実行の **stop** オプションに関する既知の問題があります。**Actions** ドロップダウンリストの **stop** オプションが期待どおりに機能せず、パイプラインの実行がキャンセルされません。

- 現在、カスタムリソース定義の変換の失敗が原因で発生する、Pipelines バージョン 1.9.x へのアップグレードに関する既知の問題があります。
  回避策: Pipelines バージョン 1.9.x にアップグレードする前に、Red Hat カスタマーポータルの solution に記載されている手順を実行してください。

4.1.5. Red Hat OpenShift Pipelines General Availability 1.8 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.8 が OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

4.1.5.1. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.8 の主な新機能について説明します。

4.1.5.1.1. Pipelines

- 今回の更新により、ARM ハードウェアで実行されている OpenShift Container Platform クラスターで Red Hat OpenShift Pipelines GA 1.8 以降を実行できるようになりました。これには、**ClusterTask** リソースと **tkn** CLI ツールのサポートが含まれます。

重要

ARM ハードウェアでの Red Hat OpenShift Pipelines の実行は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- この更新では、**TaskRun** リソースの **Step** および **Sidecar** オーバーライドが実装されています。

- この更新により、**PipelineRun** ステータス内に最小限の **TaskRun** および **Run** ステータスが追加されます。
  この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。

- 今回の更新により、パイプライン実行機能の正常な終了がアルファ機能から安定した機能に昇格されました。その結果、以前に廃止された **PipelineRunCancelled** ステータスは引き続き廃止され、将来のリリースで削除される予定です。
  この機能はデフォルトで使用できるため、**TektonConfig** カスタムリソース定義で **pipeline.enable-api-fields** フィールドを **alpha** に設定する必要がなくなりました。

- 今回の更新により、ワークスペースの名前を使用してパイプラインタスクのワークスペースを指定できるようになりました。この変更により、**Pipeline** および **PipelineTask** リソースのペアに共有ワークスペースを指定できるようになりました。ワークスペースを明示的にマップすることもできます。
  この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。

- 今回の更新により、埋め込み仕様のパラメーターが変更なしに伝播されるようになりました。

- 今回の更新により、アノテーションとラベルを使用して、**PipelineRun** リソースによって参照される **Task** リソースの必要なメタデータを指定できるようになりました。これにより、実行コンテキストに依存する **Task** メタデータは、パイプライン実行時に利用できます。

- この更新により、**params** と **results** の値にオブジェクトまたはディクショナリータイプのサポートが追加されました。この変更は後方互換性に影響し、以前のクライアントを新しい Red Hat OpenShift Pipelines バージョンで使用するなど、前方互換性を損なう場合があります。この更新により、**ArrayOrStruct** 構造が変更されます。これは、Go 言語 API をライブラリーとして使用するプロジェクトに影響します。

- この更新により、**SkippingReason** 値が **PipelineRun** ステータスフィールドの **SkippedTasks** フィールドに追加され、特定の PipelineTask がスキップされた理由をユーザーが知ることができるようになりました。

- この更新プログラムは、**Task** オブジェクトから結果を発行するために **array** 型を使用できるアルファ機能をサポートします。結果の型は **string** から **ArrayOrString** に変更されています。たとえば、タスクはタイプを指定してアレイの結果を生成できます。

```
kind: Task
apiVersion: tekton.dev/v1beta1
metadata:
  name: write-array
  annotations:
    description: |
      A simple task that writes array
spec:
  results:
    - name: array-results
      type: array
      description: The array results
...
```

さらに、タスクスクリプトを実行して、結果をアレイで入力できます。

```
$ echo -n "[\"hello\",\"world\"]" | tee $(results.array-results.path)
```

この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。

この機能は進行中であり、TEP-0076 の一部です。

4.1.5.1.2. トリガー

- この更新により、**EventListener** 仕様の **TriggerGroups** フィールドがアルファ機能から安定した機能に移行します。このフィールドを使用すると、トリガーのグループを選択および実行する前にインターセプターのセットを指定できます。
この機能はデフォルトで使用できるため、**TektonConfig** カスタムリソース定義で **pipeline.enable-api-fields** フィールドを **alpha** に設定する必要がなくなりました。

- 今回の更新により、**Trigger** リソースは、HTTPS を使用して **ClusterInterceptor** サーバーを実行することにより、エンドツーエンドの安全な接続をサポートします。

4.1.5.1.3. CLI

- 今回の更新では、**tkn taskrun export** コマンドを使用して、ライブタスクの実行をタスク

- 今回の更新では、**tkn taskrun export** コマンドを使用して、タスクの実行をクラスターから YAML ファイルにエクスポートできます。これを使用して、タスクの実行を別のクラスターにインポートできます。

- 今回の更新により、**tkn pipeline start** コマンドに **-o name** フラグを追加して、開始直後にパイプライン実行の名前を出力できるようになりました。

- 今回の更新により、利用可能なプラグインのリストが **tkn --help** コマンドの出力に追加されました。

- 今回の更新により、パイプラインの実行またはタスクの実行を削除する際に、**--keep** フラグと **--keep-since** フラグの両方を一緒に使用できるようになりました。

- 今回の更新により、非推奨の **PipelineRunCancelled** 値ではなく、**spec.status** フィールドの値として **Canceled** を使用できるようになりました。

4.1.5.1.4. Operator

- 今回の更新により、管理者はローカルの Tekton Hub インスタンスを設定して、デフォルトデータベースではなくカスタムデータベースを使用できるようになりました。

- 今回の更新では、クラスター管理者としてローカルの Tekton Hub インスタンスを有効にすると、データベースが定期的に更新され、カタログの変更が Tekton Hub Web コンソールに表示されるようになります。更新の間隔は調整できます。
以前は、カタログ内のタスクとパイプラインをデータベースに追加するために、そのタスクを手動で実行するか、cron ジョブをセットアップして実行していました。

- 今回の更新で、最小限の設定で Tekton Hub インスタンスをインストールし、実行できるようになりました。これにより、チームと連携して、必要な追加カスタマイズを決定できます。

- 今回の更新で、**GIT_SSL_CAINFO** が **git-clone** タスクに追加され、セキュアなリポジトリーをクローンできるようになりました。

4.1.5.1.5. Tekton Chains

重要

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

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

- 今回の更新により、静的トークンではなく OIDC を使用して Vault にログインすることができます。この変更により、Spire は OIDC 認証情報を生成し、信頼されるワークロードのみが vault にログインできます。また、Vault アドレスを環境変数として挿入するのではなく、設定値として渡すこともできます。

- Red Hat OpenShift Pipelines Operator を使用してインストールした場合、設定マップの直接更新はサポートされないため、**openshift-pipelines** namespace の Tekton チェーンの **chain-config** 設定マップは、Red Hat OpenShift Pipelines Operator のアップグレード後に自動的にデフォルトにリセットされます。ただし、今回の更新により、**TektonChain** カスタムリソース

を使用して Tekton Chains を設定できるようになりました。この機能により、アップグレード中に上書きされる **chain-config** 設定マップとは異なり、アップグレード後も設定を維持できます。

4.1.5.1.6. Tekton Hub

重要

Tekton Hub はテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- 今回の更新により、Operator を使用して Tekton Hub の新しいインスタンスをインストールすると、Tekton Hub ログインがデフォルトで無効になります。ログインおよび評価機能を有効にするには、Tekton Hub のインストール時に Hub API シークレットを作成する必要があります。

注記

Red Hat OpenShift Pipelines 1.7 では Tekton Hub ログインがデフォルトで有効になっているため、Operator をアップグレードすると、Red Hat OpenShift Pipelines 1.8 ではログインがデフォルトで有効になります。このログインを無効にするには、OpenShift Pipelines 1.7.x -→ 1.8.x からアップグレードした後の Tekton Hub ログインの無効化 を参照してください。

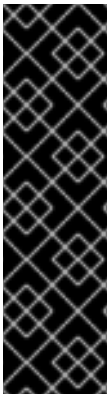
- 今回の更新により、管理者はローカルの Tekton Hub インスタンスを設定して、デフォルトデータベースではなくカスタム PostgreSQL 13 データベースを使用できるようになりました。これを行うには、**tekton-hub-db** という名前の **Secret** リソースを作成します。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: tekton-hub-db
  labels:
    app: tekton-hub-db
type: Opaque
stringData:
  POSTGRES_HOST: <hostname>
  POSTGRES_DB: <database_name>
  POSTGRES_USER: <username>
  POSTGRES_PASSWORD: <password>
  POSTGRES_PORT: <listening_port_number>
```

- 今回の更新により、カタログからデータベースにリソースを追加するために Tekton Hub Web コンソールにログインする必要がなくなりました。現在、これらのリソースは、Tekton Hub API が初めて実行を開始したときに自動的に追加されます。

- この更新プログラムは、カタログ更新 API ジョブを呼び出すことにより、30 分ごとにカタログを自動的に更新します。この間隔は user-configurable です。

4.1.5.1.7. Pipelines as Code



重要

コードとしてのパイプライン (PAC) は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- 今回の更新により、開発者は、重複したリポジトリーを Pipelines as Code run に追加しようとすると、**tkn-pac** CLI ツールから通知を受け取ります。**tkn pac create repository** を入力する場合、各リポジトリーには一意の URL が必要です。この通知は、ハイジャックエクスプロイトの防止にも役立ちます。

- 今回の更新により、開発者は新しい **tkn-pac setup cli** コマンドを使用して、Webhook メカニズムを使用して Git リポジトリーを Pipelines as Code に追加できるようになりました。このように、GitHub アプリを使用できない場合でも、Pipelines as Code を使用できます。この機能には、GitHub、GitLab、BitBucket のリポジトリーのサポートが含まれます。

- 今回の更新により、Pipelines as Code は、次のような機能を備えた GitLab 統合をサポートします。

  - プロジェクトまたはグループの ACL (アクセス制御リスト)

  - 許可されたユーザーからの **/OK-to-test** サポート

  - **/retest** サポート。

- 今回の更新により、Common Expression Language (CEL) を使用して高度なパイプラインフィルタリングを実行できます。CEL では、**PipelineRun** リソースのアノテーションを使用して、パイプラインの実行をさまざまな Git プロバイダーイベントと一致させることができます。以下に例を示します。

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
   event == "pull_request" && target_branch == "main" && source_branch == "wip"
```

- 以前は、開発者は、プルリクエストなどの Git イベントごとに **.tekton** ディレクトリーで 1 つのパイプラインしか実行できませんでした。今回の更新により、**.tekton** ディレクトリーに複数のパイプラインを実行できるようになりました。Web コンソールは、実行のステータスとレポートを表示します。パイプラインは並行して動作し、Git プロバイダーインターフェイスに報告します。

- 今回の更新により、プルリクエストで **/test** または **/retest** にコメントすることで、パイプラインの実行をテストまたは再テストできるようになりました。名前でパイプライン実行を指定することもできます。たとえば、**/test <pipelinerun_name>** または **/retest <pipelinerun-name>**

を入力できます。

- 今回の更新により、新しい **tkn-pac delete repository** コマンドを使用して、リポジトリーカスタムリソースとそれに関連付けられたシークレットを削除できるようになりました。

4.1.5.2. 互換性を失わせる変更点

- この更新により、**TaskRun** および **PipelineRun** リソースのデフォルトのメトリックレベルが次の値に変更されます。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-observability
  namespace: tekton-pipelines
  labels:
    app.kubernetes.io/instance: default
    app.kubernetes.io/part-of: tekton-pipelines
data:
  _example: |
  ...
    metrics.taskrun.level: "task"
    metrics.taskrun.duration-type: "histogram"
    metrics.pipelinerun.level: "pipeline"
    metrics.pipelinerun.duration-type: "histogram"
```

- 今回の更新により、アノテーションまたはラベルが **Pipeline** および **PipelineRun** リソースの両方にある場合、**Run** タイプの値が優先されます。アノテーションまたはラベルが **Task** および **TaskRun** リソースにある場合も同様です。

- Red Hat OpenShift Pipelines 1.8 では、以前に非推奨の **PipelineRun.Spec.ServiceAccountNames** フィールドが削除されました。代わりに **PipelineRun.Spec.TaskRunSpecs** フィールドを使用してください。

- Red Hat OpenShift Pipelines 1.8 では、以前に非推奨の **TaskRun.Status.ResourceResults.ResourceRef** フィールドが削除されました。代わりに **TaskRun.Status.ResourceResults.ResourceName** フィールドを使用してください。

- Red Hat OpenShift Pipelines 1.8 では、以前に非推奨となった **Conditions** リソースタイプが削除されました。**Conditions** リソースを、これが含まれる **Pipeline** リソース定義から削除します。代わりに **PipelineRun** 定義で **when** 式を使用してください。

- Tekton Chains では、**tekton-provenance** 形式は本リリースで削除されました。代わりに、**TektonChain** カスタムリソースで **"artifacts.taskrun.format": "in-toto"** を設定して、**in-toto** 形式を使用します。

- Pipeline が Code 0.5.x として同梱される Red Hat OpenShift Pipelines 1.7.x。現在の更新には、Pipeline が Code 0.10.x として同梱されます。この変更により、新規コントローラーの **openshift-pipelines** namespace に新規ルートが作成されます。このルートは、Pipeline を Code として使用する GitHub Apps または Webhook で更新する必要があります。ルートを取得するには、以下のコマンドを使用します。

```
$ oc get route -n openshift-pipelines pipelines-as-code-controller \
  --template='https://{{ .spec.host }}'
```

- 今回の更新で、コードとしてのパイプラインは、**Repository** カスタムリソース定義 (CRD) のデフォルトの秘密鍵の名前を変更します。CRD で、**token** を **provider.token** に置き換え、**secret** を **webhook.secret** に置き換えます。

- 今回の更新で、Pipelines as Code は、特別なテンプレート変数を、プライベートリポジトリーの複数のパイプライン実行をサポートするものに置き換えます。パイプラインの実行で、**secret: pac-git-basic-auth-{{repo_owner}}-{{repo_name}}** を **secret: {{ git_auth_secret }}** に置き換えます。

- 今回の更新により、コードとしての Pipeline が **tkn-pac** CLI ツールで以下のコマンドを更新するようになりました。

  - **tkn pac repository create** を **tkn pac create repository** に置き換えます。

  - **tkn pac repository delete** を **tkn pac delete repository** に置き換えます。

  - **tkn pac repository list** を **tkn pac list** に置き換えます。

4.1.5.3. 非推奨および削除された機能

- OpenShift Container Platform 4.11 以降、Red Hat OpenShift Pipelines Operator をインストールおよびアップグレードするための **preview** および **stable** チャネルは削除されています。Operator をインストールしてアップグレードするには、適切な **pipelines-<version>** チャネル、または最新の安定バージョンの **latest** チャネルを使用します。たとえば、Pipelines Operator バージョン **1.8.x** をインストールするには、**pipelines-1.8** チャネルを使用します。

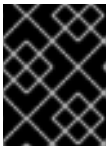
  

  注記

  OpenShift Container Platform 4.10 以前のバージョンでは、**preview** および **stable** チャンネルを使用して Operator をインストールおよびアップグレードできます。

- Red Hat OpenShift Pipelines GA 1.6 で廃止された **tekton.dev/v1alpha1** API バージョンのサポートは、今後の Red Hat OpenShift Pipelines GA 1.9 リリースで削除される予定です。この変更は、**TaskRun**、**PipelineRun**、**Task**、**Pipeline**、および同様の **tekton.dev/v1alpha1** リソースを含むパイプラインコンポーネントに影響します。別の方法として、 Migrating From Tekton v1alpha1 to Tekton v1beta1 で説明されているように、既存のリソースを **apiVersion: tekton.dev/v1beta1** を使用するように更新します。

  tekton.dev/v1alpha1 API バージョンのバグ修正とサポートは、現在の GA 1.8 ライフサイクルの終了までのみ提供されます。

  重要

  Tekton Operator の場合、**operator.tekton.dev/v1alpha1** API バージョンは **非推奨ではありません。この値を変更する必要はありません。**

- Red Hat OpenShift Pipelines 1.8 では、**PipelineResource** カスタムリソース (CR) が利用可能ですが、サポートされなくなりました。**PipelineResource** CR は Tech Preview 機能であり、**tekton.dev/v1alpha1** API の一部であり、廃止予定であり、今後の Red Hat OpenShift Pipelines GA 1.9 リリースで削除される予定でした。

- Red Hat OpenShift Pipelines 1.8 では、**Condition** カスタムリソース (CR) が削除されています。**Condition** CR は **tekton.dev/v1alpha1** API の一部でしたが、これは非推奨であり、今後の Red Hat OpenShift Pipelines GA 1.9 リリースで削除される予定です。

- Red Hat OpenShift Pipelines 1.8 では、**gsutil** の **gcr.io** イメージが削除されました。この削除により、このイメージに依存する **Pipeline** リソースを含むクラスターが壊れる可能性があります。バグ修正とサポートは、Red Hat OpenShift Pipelines 1.7 ライフサイクルが終了するまでのみ提供されます。

- Red Hat OpenShift Pipelines 1.8 では、**PipelineRun.Status.TaskRuns** および **PipelineRun.Status.Runs** フィールドは非推奨となり、将来のリリースで削除される予定です。TEP-0100: PipelineRuns に埋め込まれた TaskRuns と Runs Status を参照してください。

- Red Hat OpenShift Pipelines 1.8 では、**pipelineRunCancelled** 状態は非推奨となり、今後のリリースで削除される予定です。**PipelineRun** オブジェクトの正常な終了は、アルファ機能から安定した機能にプロモートされるようになりました。(TEP-0058: パイプライン実行の正常な終了 を参照してください。) 別の方法として、**Cancelled** 状態を使用できます。これは **pipelineRunCancelled** 状態を置き換えます。
Pipeline および **Task** リソースを変更する必要はありません。パイプラインの実行をキャンセルするツールがある場合は、次のリリースでツールを更新する必要があります。この変更は、CLI、IDE 拡張機能などのツールにも影響を与え、新しい **PipelineRun** ステータスをサポートするようにします。

  この機能はデフォルトで使用できるため、**TektonConfig** カスタムリソース定義で **pipeline.enable-api-fields** フィールドを **alpha** に設定する必要がなくなりました。

- Red Hat OpenShift Pipelines 1.8 では、**PipelineRun** の **timeout** フィールドが非推奨になりました。代わりに、**PipelineRun.Timeouts** フィールドを使用してください。これは現在、アルファ機能から安定した機能に昇格しています。
この機能はデフォルトで使用できるため、**TektonConfig** カスタムリソース定義で **pipeline.enable-api-fields** フィールドを **alpha** に設定する必要がなくなりました。

- Red Hat OpenShift Pipelines 1.8 では、**init** コンテナーは **LimitRange** オブジェクトのデフォルトのリクエスト計算から省略されています。

4.1.5.4. 既知の問題

- **s2i-nodejs** パイプラインは、**nodejs:14-ubi8-minimal** イメージストリームを使用して、source-to-image (S2I) ビルドを実行できません。そのイメージストリームを使用すると **error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127** メッセージが生成されます。
回避策: **nodejs:14-ubi8-minimal** イメージストリームではなく、**nodejs:14-ubi8** を使用します。

- Maven および Jib Maven クラスタータスクを実行する場合には、デフォルトのコンテナーイメージは Intel(x86) アーキテクチャーでのみサポートされます。したがって、タスクは ARM、IBM Power Systems (ppc64le)、IBM Z、および LinuxONE (s390x) クラスターで失敗します。
回避策: **MAVEN_IMAGE** パラメーター値を **maven:3.6.3-adoptopenjdk-11** に設定して、カスタムイメージを指定します。

  ヒント

  tkn hub を使用して、ARM、IBM Power Systems (ppc64le)、IBM Z、および LinuxONE (s390x) に Tekton カタログに基づくタスクをインストールする前に、これらのプラットフォームでタスクを実行できるかどうかを確認してください。**ppc64le** および **s390x** がタスク情報の Platforms セクションに一覧表示されているかどうかを確認するには、**tkn hub info task <name>** コマンドを実行します。

- ARM、IBM Power Systems、IBM Z、および LinuxONE では、**s2i-dotnet** クラスタータスクはサポートされていません。

- 暗黙的なパラメーターマッピングは、最上位の **Pipeline** または **PipelineRun** 定義から **taskRef** タスクにパラメーターを誤って渡します。マッピングは、トップレベルのリソースからインライン **taskSpec** 仕様のタスクにのみ行う必要があります。この問題は、**TektonConfig** カスタムリソース定義の **pipeline** セクションで **enable-api-fields** フィールドを **alpha** に設定することにより、この機能が有効になっているクラスターにのみ影響します。

4.1.5.5. 修正された問題

- この更新の前は、Web コンソールの開発者ビューでのパイプライン実行のメトリックは不完全で古くなっていました。今回の更新で問題が修正され、指標が正しくなりました。

- この更新の前は、パイプラインに失敗した 2 つの並列タスクがあり、そのうちの 1 つが **retries=2** であった場合、最後のタスクは実行されず、パイプラインはタイムアウトして実行に失敗しました。たとえば、**pipelines-operator-subscription** タスクが次のエラーメッセージで断続的に失敗しました。**Unable to connect to the server: EOF**。今回の更新で、最終タスクが常に実行されるように問題が修正されました。

- この更新の前は、タスクの実行が失敗したためにパイプラインの実行が停止した場合、他のタスクの実行が再試行を完了しない可能性がありました。その結果、**finally** タスクがスケジュールされず、パイプラインがハングしました。 今回の更新で問題が解決されました。**TaskRuns** および **Run** オブジェクトは、パイプラインの実行が停止したときに (正常な停止によっても) 再試行できるため、パイプラインの実行を完了できます。

- この更新により、**TaskRun** オブジェクトが存在する namespace に 1 つ以上の **LimitRange** オブジェクトが存在する場合のリソース要件の計算方法が変更されます。スケジューラーは、**step** コンテナーを考慮し、**LimitRange** オブジェクトからの要求を因数分解するときに、サイドカーコンテナーなどの他のすべてのアプリコンテナーを除外するようになりました。

- この更新の前は、特定の条件下で、フラグパッケージが二重ダッシュフラグターミネータ **--** の直後のサブコマンドを誤って解析する場合がありました。その場合、実際のコマンドではなく、エントリーポイントサブコマンドが実行されました。今回の更新では、このフラグ解析の問題が修正され、エントリーポイントが正しいコマンドを実行できるようになりました。

- この更新の前は、イメージのプルが失敗した場合、またはそのプルステータスが不完全であった場合、コントローラーが複数のパニックを生成する可能性がありました。この更新では、**status.TaskSpec** 値ではなく **step.ImageID** 値をチェックすることで問題が修正されています。

- この更新の前は、スケジュールされていないカスタムタスクを含むパイプラインの実行をキャンセルすると、**PipelineRunCouldntCancel** エラーが発生していました。今回の更新でこの問題が修正されています。エラーを生成することなく、スケジュールされていないカスタムタスクを含むパイプラインの実行をキャンセルできます。

- この更新の前は、**$params["<NAME>"]** または **$params['<NAME>']** の **<NAME>** にドット文字 (**.**) が含まれている場合、ドットの右側の名前のどの部分も含まれていませんでした。たとえば、**$params["org.ipsum.lorem"]** から、**org** のみが抽出されました。
今回の更新で問題が修正され、**$params** が完全な値を取得するようになりました。たとえば、**$params["org.ipsum.lorem"]** と **$params['org.ipsum.lorem']** は有効で、**<NAME>** の値全体である **org.ipsum.lorem** が抽出されます。

  <NAME> が一重引用符または二重引用符で囲まれていない場合にも、エラーが出力されます。たとえば、**$params.org.ipsum.lorem** は有効ではなく、検証エラーが発生します。

- 今回の更新により、**Trigger** リソースはカスタムインターセプターをサポートし、カスタムインターセプターサービスのポートが **ClusterInterceptor** 定義ファイルのポートと同じになるようにします。

- この更新の前は、Tekton Chains および Operator コンポーネントの **tkn version** コマンドが正しく機能しませんでした。今回の更新で問題が修正され、コマンドが正しく機能し、それらのコンポーネントのバージョン情報が返されるようになりました。

- この更新の前に、**tkn pr delete --ignore-running** コマンドを実行し、パイプラインの実行に **status.condition** 値がない場合、**tkn** CLI ツールは null-pointer エラー (NPE) を生成しました。今回の更新で問題が修正され、CLI ツールがエラーを生成し、実行中のパイプライン実行を正しく無視するようになりました。

- この更新の前に、**tkn pr delete --keep <value>** または **tkn tr delete --keep <value>** コマンドを使用し、パイプラインの実行またはタスクの実行の数が値よりも少ない場合、コマンドは予想通りのエラー。今回の更新で問題が修正され、これらの条件下でコマンドが正しくエラーを返すようになりました。

- この更新の前に、**-p** または **-t** フラグと **--ignore-running** フラグを指定して **tkn pr delete** または **tkn tr delete** コマンドを使用した場合、コマンドは実行中または保留中のリソースを誤って削除しました。今回の更新で問題が修正され、これらのコマンドが実行中または保留中のリソースを正しく無視するようになりました。

- 今回の更新により、**TektonChain** カスタムリソースを使用して Tekton Chains を設定できるようになりました。この機能により、アップグレード中に上書きされる **chain-config** 設定マップとは異なり、アップグレード後も設定を維持できます。

- 今回の更新により、**buildah** および **s2i** クラスタータスクを除き、**ClusterTask** リソースはデフォルトで root として実行されなくなりました。

- 今回の更新前は、最初の引数として **init** を使用し、その後に 2 つ以上の引数を使用すると、Red Hat OpenShift Pipelines 1.7.1 でのタスクが失敗していました。今回の更新により、フラグが正しく解析され、タスクが正常に実行されるようになりました。

- 今回の更新以前は、無効なロールバインディングにより、OpenShift Container Platform 4.9 および 4.10 への Red Hat OpenShift Pipelines Operator のインストールは、以下のエラーメッセージと共に失敗していました。

  ```
  error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
  RoleBinding.rbac.authorization.k8s.io
  "openshift-operators-prometheus-k8s-read-binding" is invalid:
  roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io", Kind:"Role",
  Name:"openshift-operator-read"}: cannot change roleRef
  ```

  今回の更新で問題が修正され、障害が発生しなくなりました。

- 以前は、Red Hat OpenShift Pipelines Operator をアップグレードすると **pipeline** サービスアカウントが再作成され、サービスアカウントにリンクされたシークレットが失われていました。今回の更新でこの問題が修正されています。アップグレード中に、Operator は **pipeline** サービスアカウントを再作成しなくなりました。その結果、**pipeline** サービスアカウントにアタッチされたシークレットはアップグレード後も保持され、リソース (タスクとパイプライン) は引き続き正しく機能します。

- 今回の更新により、**TektonConfig** カスタムリソース (CR) でインフラストラクチャーノード設定が設定されている場合、Pipelines as Code Pod はインフラストラクチャーノードで実行されます。

- 以前は、リソースプルーナーを使用して、各 namespace Operator が個別のコンテナーで実行されるコマンドを作成していました。この設計は、namespaces の数が多いクラスターで大量のリソースを消費しました。たとえば、1 つのコマンドを実行するために、1000 個の namespace を持つクラスターは、Pod 内に 1000 個のコンテナーを生成しました。
今回の更新でこの問題が修正されています。すべてのコマンドがループ内の 1 つのコンテナーで実行されるように、namespace ベースの設定をジョブに渡します。

- Tekton Chains では、**signing-secrets** と呼ばれるシークレットを定義して、タスクとイメージの署名に使用されるキーを保持する必要があります。ただし、この更新の前に、Red Hat OpenShift Pipelines Operator を更新すると、このシークレットがリセットまたは上書きされ、キーが失われました。今回の更新でこの問題が修正されています。これで、Operator を介して Tekton Chains をインストールした後にシークレットが設定された場合、シークレットは保持され、アップグレードによって上書きされなくなりました。

- 今回の更新以前は、すべての S2I ビルドタスクが以下の様なエラーメッセージと共に失敗していました。

```
Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
operation not permitted
time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
/proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
```

  今回の更新により、**pipelines-scc** セキュリティーコンテキスト制約 (SCC) は、 **Buildah** および **S2I** クラスタータスクに必要な **SETFCAP** 機能と互換性が確保されています。その結果、**Buildah** および **S2I** ビルドタスクを正常に実行できます。

  さまざまな言語やフレームワークで書かれたアプリケーションに対して **Buildah** クラスタータスクおよび **S2I** ビルドタスクを正常に実行するには、**build** や **push** などの適切な **steps** オブジェクトに以下のスニペットを追加します。

```
securityContext:
  capabilities:
    add: ["SETFCAP"]
```

- 今回の更新前は、Red Hat OpenShift Pipelines Operator のインストールに予想以上に時間がかかりました。この更新プログラムは、インストールプロセスを高速化するために一部の設定を最適化します。

- 今回の更新により、Buildah および S2I クラスタータスクの手順が以前のバージョンよりも少なくなりました。一部のステップは 1 つのステップに結合されているため、**ResourceQuota** および **LimitRange** オブジェクトでより適切に機能し、必要以上のリソースを必要としません。

- この更新により、クラスタータスクの Buildah、**tkn** CLI ツール、および **skopeo** CLI ツールのバージョンがアップグレードされます。

- 今回の更新前は、いずれかの namespace が **Terminating** 状態の場合、RBAC リソースの作成時に Operator が失敗していました。今回の更新により、Operator は **Terminating** 状態の namespace を無視し、RBAC リソースを作成します。

- この更新の前は、予想どおり、prune cronjobs の Pod がインフラストラクチャーノードでスケジュールされていませんでした。代わりに、それらはワーカーノードでスケジュールされているか、まったくスケジュールされていませんでした。今回の更新により、**TektonConfig** カスタムリソース (CR) で設定されている場合、これらのタイプの Pod をインフラストラクチャーノードでスケジュールできるようになりました。

4.1.5.6. Red Hat OpenShift Pipelines General Availability (GA) 1.8.1 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.8.1 が OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

4.1.5.6.1. 既知の問題

- デフォルトでは、セキュリティーを強化するために、コンテナーのアクセス権が制限されています。制限付きのアクセス許可は、Red Hat OpenShift Pipelines Operator のすべてのコントローラー Pod と、一部のクラスタータスクに適用されます。アクセス権が制限されているため、特定の設定では **git-clone** クラスタータスクが失敗します。
  回避策: なし。問題 SRVKP-2634 を確認してください。

- インストーラーセットが失敗した状態の場合、**TektonConfig** カスタムリソースのステータスが **False** ではなく **True** として誤表示されます。

  例: 失敗したインストーラセット

```
$ oc get tektoninstallerset
NAME                                READY   REASON
addon-clustertasks-nx5xz            False   Error
addon-communityclustertasks-cfb2p   True
addon-consolecli-ftrb8              True
addon-openshift-67dj2               True
addon-pac-cf7pz                     True
addon-pipelines-fvllm               True
addon-triggers-b2wtt                True
addon-versioned-clustertasks-1-8-hqhnw   False   Error
pipeline-w75ww                      True
postpipeline-lrs22                  True
prepipeline-ldlhw                   True
rhosp-rbac-4dmgb                    True
trigger-hfg64                       True
validating-mutating-webhoook-28rf7   True
```

  例: 正しくない TektonConfig ステータス

```
$ oc get tektonconfig config
NAME     VERSION   READY   REASON
config   1.8.1     True
```

4.1.5.6.2. 修正された問題

- この更新まで、プルーナーは実行中のパイプラインのタスク実行を削除し、警告 **some tasks were indicated completed without ancestors being done** を表示していました。今回の更新により、プルーナーは、実行中のパイプラインの一部であるタスク実行を保持します。

- この更新まで、**pipeline-1.8** が Red Hat OpenShift Pipelines Operator 1.8.x をインストールするためのデフォルトのチャネルでした。今回の更新により、**latest** がデフォルトのチャネルになりました。

- この更新まで、コードとしてのパイプラインのコントローラー Pod は、ユーザーによって公開された証明書にアクセスできませんでした。今回の更新により、コードとしてのパイプラインは、自己署名証明書またはカスタム証明書によって保護されたルートと Git リポジトリーにアクセスできるようになりました。

- この更新まで、Red Hat OpenShift Pipelines 1.7.2 から 1.8.0 にアップグレードすると、タスクが RBAC エラーで失敗していました。今回の更新により、タスクは RBAC エラーなしで正常に実行されます。

- この更新まで、**tkn** CLI ツールを使用して、**array** 型の **result** オブジェクトを含むタスク実行とパイプライン実行を削除できませんでした。今回の更新により、**tkn** CLI ツールを使用して、**array** 型の **result** オブジェクトを含むタスク実行とパイプライン実行を削除できます。

- この更新まで、パイプライン仕様に **array** 型の **ENV_VARS** パラメーターを持つタスクが含まれていた場合、パイプラインの実行は **invalid input params for task func-buildpacks: param types don't match the user-specified type: [ENV_VARS]** エラーで失敗していました。今回の更新により、そのようなパイプラインおよびタスク仕様でのパイプライン実行は失敗しなくなりました。

- この更新まで、クラスター管理者は、コンテナーレジストリーにアクセスするための **Buildah** クラスタータスクに **config.json** ファイルを提供できませんでした。今回の更新により、クラスター管理者は、**dockerconfig** ワークスペースを使用して、**Buildah** クラスタータスクに **config.json** ファイルを提供できるようになりました。

4.1.5.7. Red Hat OpenShift Pipelines General Availability (GA) 1.8.2 のリリースノート

今回の更新により、Red Hat OpenShift Pipelines General Availability (GA) 1.8.2 が OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

4.1.5.7.1. 修正された問題

- この更新の前は、SSH キーを使用してリポジトリーのクローンを作成すると、**git-clone** タスクが失敗していました。今回の更新により、**git-init** タスクでの root 以外のユーザーのロールが削除され、SSH プログラムは **$HOME/.ssh/** ディレクトリーで正しいキーを検索します。

4.1.6. Red Hat OpenShift Pipelines General Availability 1.7 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.7 が OpenShift Container Platform 4.9、4.10、および 4.11 で利用可能になりました。

4.1.6.1. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.7 の主な新機能について説明します。

4.1.6.1.1. Pipelines

- 今回の更新では、**pipelines-<version>** が Red Hat OpenShift Pipelines Operator をインストールするためのデフォルトのチャネルです。たとえば、Pipelines Operator バージョン **1.7** をインストールするためのデフォルトのチャネルは **pipelines-1.7** です。クラスター管理者は、**latest** チャネルを使用して、Operator の最新の安定バージョンをインストールすることもできます。

注記

preview チャネルと **stable** チャネルは廃止され、将来のリリースで削除される予定です。

- ユーザー namespace でコマンドを実行すると、コンテナーは **root** (ユーザー ID **0**) として実行されますが、ホストに対するユーザー特権があります。この更新では、ユーザー namespace で pod を実行するには、CRI-O が期待するアノテーションを渡す必要があります。
    - すべてのユーザーにこれらのアノテーションを追加するには、**oc edit clustertask buildah** コマンドを実行し、**buildah** クラスタータスクを編集します。
    - 特定の namespace にアノテーションを追加するには、クラスタータスクをタスクとしてその namespace にエクスポートします。
- この更新の前は、特定の条件が満たされない場合、**when** 式は **Task** オブジェクトとその依存タスクをスキップしていました。今回の更新により、**when** 式のスコープを設定して、従属タスクではなく、**Task** オブジェクトのみを保護できるようになりました。この更新を有効にするには、**TektonConfig** CRD で **scope-when-expressions-to-task** フラグを **true** に設定します。

注記

scope-when-expressions-to-task フラグは非推奨であり、将来のリリースで削除される予定です。パイプラインのベストプラクティスとして、保護された **Task** のみを対象とする式の **when** に使用します。

- この更新では、タスク内のワークスペースの **subPath** フィールドで変数置換を使用できます。
- 今回の更新では、一重引用符または二重引用符を含む角かっこ表記を使用して、パラメーターと結果を参照できます。この更新以前は、ドット表記しか使用できませんでした。たとえば、次は同等になりました。
    - **$(param.myparam)**、**$(param['myparam'])**、および **$(param["myparam"])**。
      一重引用符または二重引用符を使用して、**"."** などの問題のある文字を含むパラメーター名を囲むことができます。たとえば、**$(param['my.param'])** と **$(param["my.param"])**。
- この更新により、**enable-api-fields** フラグを有効にせずに、タスク定義にステップの **onError** パラメーターを含めることができます。

4.1.6.1.2. トリガー

- この更新により、**feature-flag-triggers** 設定マップに新しいフィールド **labels-exclusion-pattern** が追加されました。このフィールドの値を正規表現 (regex) パターンに設定できます。コントローラーは、正規表現パターンに一致するラベルを、イベントリスナーからイベントリスナー用に作成されたリソースへの伝播から除外します。
- この更新により、**TriggerGroups** フィールドが **EventListener** 仕様に追加されました。このフィールドを使用すると、トリガーのグループを選択して実行する前に実行するインターセプターのセットを指定できます。この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。
- この更新により、**Trigger** リソースは、**TriggerTemplate** テンプレートによって定義されたカスタム実行をサポートします。
- この更新により、トリガーは **EventListener** Pod からの Kubernetes イベントの生成をサポートします。
- この更新により、次のオブジェクトのカウントメトリックが使用可能になります: **ClusterInteceptor**、**EventListener**、**TriggerTemplate**、**ClusterTriggerBinding**、および **TriggerBinding**。

- この更新により、**ServicePort** 仕様が Kubernetes リソースに追加されます。この仕様を使用して、イベントリスナーサービスを公開するポートを変更できます。デフォルトのポートは **8080** です。

- この更新では、**EventListener** 仕様の **targetURI** フィールドを使用して、トリガー処理中にクラウドイベントを送信できます。この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。

- この更新により、**tekton-triggers-eventlistener-roles** オブジェクトには、既存の **create** 動詞に加えて、**patch** 動詞が含まれるようになりました。

- この更新により、**securityContext.runAsUser** パラメーターがイベントリスナーのデプロイメントから削除されます。

4.1.6.1.3. CLI

- この更新では、**tkn [pipeline | pipelinerun] export** コマンドは、パイプラインまたはパイプライン実行を YAML ファイルとしてエクスポートします。以下に例を示します。

  - **openshift-pipelines** namespace に **test_pipeline** という名前のパイプラインをエクスポートします。

    ```
    $ tkn pipeline export test_pipeline -n openshift-pipelines
    ```

  - **openshift-pipelines** namespace に **test_pipeline_run** という名前のパイプランランをエクスポートします。

    ```
    $ tkn pipelinerun export test_pipeline_run -n openshift-pipelines
    ```

- この更新により、**--grace** オプションが **tkn pipelinerun cancel** に追加されます。**--grace** オプションを使用して、パイプラインの実行を強制的に終了するのではなく、適切に終了します。この機能を有効にするには、**TektonConfig** カスタムリソース定義の **パイプライン** セクションで、**enable-api-fields** フィールドを **alpha** に設定する必要があります。

- この更新により、Operator バージョンと Chains バージョンが **tkn version** コマンドの出力に追加されます。

重要

Tekton Chains はテクノロジープレビュー機能です。

- この更新により、パイプラインの実行をキャンセルすると、**tkn pipelinerun describe** コマンドはキャンセルされたすべてのタスクの実行を表示します。この修正以前は、1 つのタスク実行のみが表示されていました。

- この更新により、**tkn [t | p | ct] start** コマンドのスキップを **--skip-optional-workspace** フラグで実行したときに、オプションのワークスペースの要求仕様を省略できるようになりました。インタラクティブモードで実行している場合はスキップすることもできます。

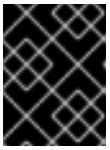
- この更新では、**tkn chains** コマンドを使用して Tekton Chains を管理できます。**--chains-namespace** オプションを使用し Tekton Chains をインストールする namespace を指定することもできます。

重要

Tekton Chains はテクノロジープレビュー機能です。

4.1.6.1.4. Operator

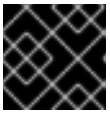
- この更新では、Red Hat OpenShift Pipelines Operator を使用して、Tekton Hub および Tekton Chains をインストールおよびデプロイできます。



重要

Tekton Chains とクラスターへの Tekton Hub のデプロイメントは、テクノロジープレビュー機能です。

- この更新により、アドオンオプションとして Pipelines as Code (PAC) を見つけて使用できるようになります。



重要

Pipelines as Code は、テクノロジープレビュー機能です。

- この更新により、**communityClusterTasks** パラメーターを **false** に設定することにより、コミュニティークラスタータスクのインストールを無効にできるようになりました。以下に例を示します。

```
...
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "true"
    - name: communityClusterTasks
      value: "false"
...
```

- この更新では、**TektonConfig** カスタムリソースの **enable-devconsole-integration** フラグを **false** に設定することで、Tekton Hub と **Developer** パースペクティブの統合を無効にできます。以下に例を示します。

```
...
hub:
  params:
    - name: enable-devconsole-integration
      value: "true"
...
```

- 今回の更新により、**operator-config.yaml** 設定マップにより、**tkn version** コマンドの出力で Operator バージョンを表示できるようになります。

- この更新により、**argocd-task-sync-and-wait** タスクのバージョンが **v0.2** に変更されます。

- この **TektonConfig**CRD の更新により、**oc get tektonconfig** コマンドは OPerator のバージョンを表示します。
- この更新により、サービスモニターがトリガーメトリックに追加されます。

4.1.6.1.5. ハブ

重要

Tekton Hub をクラスターにデプロイすることは、テクノロジープレビュー機能です。

Tekton Hub は、CI/CD ワークフローの再利用可能なタスクとパイプラインを検出、検索、および共有するのに役立ちます。Tekton Hub のパブリックインスタンスは、hub.tekton.dev で利用できます。

Red Hat OpenShift Pipelines 1.7 を確認しながら、クラスター管理者は Tekton Hub のカスタムインスタンスをエンタープライズクラスターにインストールしてデプロイすることもできます。組織に固有の再利用可能なタスクとパイプラインを使用してカタログをキュレートできます。

4.1.6.1.6. チェーン

重要

Tekton Chains はテクノロジープレビュー機能です。

Tekton Chains は、Kubernetes カスタムリソース定義 (CRD) コントローラーです。これを使用して、Red Hat OpenShift Pipelines を使用して作成されたタスクおよびパイプラインのサプライチェーンセキュリティーを管理できます。

デフォルトでは、Tekton Chains は OpenShift Container Platform クラスターで実行されるタスクをモニターします。Chains は、完了したタスク実行のスナップショットを取得し、それらを 1 つ以上の標準ペイロード形式に変換し、すべてのアーティファクトに署名して保存します。

Tekton Chains は、次の機能をサポートしています。

- 暗号化キータイプと **cosign** などのサービスを使用して、タスク実行、タスク実行結果、および OCI レジストリーイメージに署名できます。
- **in-toto** などの認証形式を使用できます。
- OCI リポジトリーをストレージバックエンドとして使用して、署名と署名されたアーティファクトを安全に保存できます。

4.1.6.1.7. Pipelines as Code (PAC)

重要

Pipelines as Code は、テクノロジープレビュー機能です。

Pipelines as Code を使用すると、クラスター管理者と必要な権限を持つユーザーは、パイプラインテンプレートをソースコード Git リポジトリーの一部として定義できます。設定された Git リポジトリーのソースコードプッシュまたはプルリクエストによってトリガーされると、この機能はパイプラインを実行し、ステータスを報告します。

Pipelines as Code は、次の機能をサポートしています。

- プルリクエストのステータス。プルリクエストを反復処理する場合、プルリクエストのステータスと制御は Git リポジトリーをホストしているプラットフォームで実行されます。
- GitHub は API をチェックして、再チェックを含むパイプライン実行のステータスを設定します。
- GitHub のプルリクエストとコミットイベント。
- **/retest** などのコメントでリクエストアクションをプルします。
- Git イベントのフィルタリング、およびイベントごとの個別のパイプライン。
- ローカルタスク、Tekton Hub、およびリモート URL のパイプラインでの自動タスク解決。
- 設定を取得するための GitHub blobs およびオブジェクト API の使用。
- GitHub 組織を介して、または Prow スタイルの **OWNER** ファイルを使用したアクセス制御リスト (ACL)。
- **tkn** CLI ツール用の **tkn pac** プラグイン。これを使用して Pipelines as Code リポジトリーとブートストラップを管理できます。
- GitHub アプリケーション、GitHub Webhook、Bitbucket Server、および Bitbucket Cloud のサポート。

4.1.6.2. 非推奨の機能

- 重大な変更: この更新により、**TektonConfig** カスタムリソース (CR) から **disable-working-directory-overwrite** および **disable-home-env-overwrite** フィールドが削除されます。その結果、**TektonConfig** CR は **$HOME** 環境変数と **workingDir** パラメーターを自動的に設定しなくなりました。**タスク** カスタムリソース定義 (CRD) の **env** および **workingDir** フィールドを使用して、引き続き **$HOME** 環境変数と **workingDir** パラメーターを設定できます。
- **Conditions** カスタムリソース定義 (CRD) タイプは非推奨であり、将来のリリースで削除される予定です。代わりに、推奨される **When** 式を使用してください。
- 重大な変更: **EventListener** と **TriggerBinding** の値を指定しない場合、**Triggers** リソースはテンプレートを検証し、エラーを生成します。

4.1.6.3. 既知の問題

- Maven および Jib Maven クラスタータスクを実行する場合には、デフォルトのコンテナーイメージは Intel(x86) アーキテクチャーでのみサポートされます。したがって、タスクは ARM、IBM Power Systems (ppc64le)、IBM Z、および LinuxONE (s390x) クラスターで失敗します。回避策として、**MAVEN_IMAGE** パラメーターの値を **maven:3.6.3-adoptopenjdk-11** に設定すると、カスタムイメージを指定できます。

  ヒント

  tkn hub を使用して、ARM、IBM Power Systems (ppc64le)、IBM Z、および LinuxONE (s390x) に Tekton カタログに基づくタスクをインストールする前に、これらのプラットフォームでタスクを実行できるかどうかを確認してください。**ppc64le** および **s390x** がタスク情報の Platforms セクションに一覧表示されているかどうかを確認するには、**tkn hub info task <name>** コマンドを実行します。

- IBM Power Systems、IBM Z、および LinuxONE では、**s2i-dotnet** クラスタータスクはサポートされません。
- **nodejs:14-ubi8-minimal** イメージストリームを使用すると、以下のエラーが生成されるため、使用できません。

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

- 暗黙的なパラメーターマッピングは、最上位の **Pipeline** または **PipelineRun** 定義から **taskRef** タスクにパラメーターを誤って渡します。マッピングは、トップレベルのリソースからインライン **taskSpec** 仕様のタスクにのみ行う必要があります。この問題は、**TektonConfig** カスタムリソース定義の **pipeline** セクションで **enable-api-fields** フィールドを **alpha** に設定することにより、この機能が有効になっているクラスターにのみ影響します。

4.1.6.4. 修正された問題

- 今回の更新では、**labels** や **annotations** などのメタデータが **Pipeline** オブジェクト定義と **PipelineRun** オブジェクト定義の両方に存在する場合、**PipelineRun** タイプの値が優先されます。**Task** オブジェクトと **TaskRun** オブジェクトで同様の動作が見られます。
- この更新では、**timeouts.tasks** フィールドまたは **timeouts.finally** フィールドが **0** に設定されている場合、**timeouts.pipeline** も **0** に設定されます。
- この更新により、シバンを使用しないスクリプトから **-x** セットフラグが削除されました。この修正により、スクリプト実行による潜在的なデータ漏洩が減少します。
- この更新により、Git クレデンシャルのユーザー名に存在するバックスラッシュ文字は、**.gitconfig** ファイルの追加のバックスラッシュでエスケープされます。
- この更新により、**EventListener** オブジェクトの **finalizer** プロパティーは、ロギングおよび設定マップのクリーンアップに必要なくなりました。
- この更新により、イベントリスナーサーバーに関連付けられているデフォルトの HTTP クライアントが削除され、カスタム HTTP クライアントが追加されます。その結果、タイムアウトが改善されました。
- この更新により、トリガークラスターのロールが所有者の参照で機能するようになりました。
- この更新では、複数のインターセプターが拡張機能を返す場合、イベントリスナーの競合状態は発生しません。
- この更新により、**tkn pr delete** コマンドは、**ignore-running** フラグで実行されているパイプラインを削除しません。
- この更新では、アドオンパラメーターを変更しても、Operator Pod は再起動し続けません。
- この更新により、サブスクリプションおよび設定カスタムリソースで設定されていない場合、**tkn serve** CLI Pod はインフラノードでスケジュールされます。
- この更新では、指定されたバージョンのクラスタータスクはアップグレード中に削除されません。

4.1.6.5. Red Hat OpenShift Pipelines General Availability 1.7.1 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.7.1 が OpenShift Container Platform 4.9、4.10、および 4.11 で利用可能になりました。

4.1.6.5.1. 修正された問題

- 今回の更新以前は、Red Hat OpenShift Pipelines Operator をアップグレードすると、Tekton Hub に関連付けられたデータベースのデータが削除され、新規データベースがインストールされていました。今回の更新により、Operator のアップグレードでデータが保存されるようになりました。

- 今回の更新以前は、クラスター管理者のみが OpenShift Container Platform コンソールでパイプラインメトリックにアクセスできていました。今回の更新により、他のクラスターロールを持つユーザーもパイプラインメトリックにアクセスできるようになりました。

- 今回の更新以前は、大量の終了メッセージを生成するタスクが含まれるパイプラインの場合、パイプラインの実行に失敗しました。Pod 内のすべてのコンテナーの終了メッセージの合計サイズは 12 KB を超えることができないために、パイプライン実行が失敗しました。今回の更新により、同じイメージを使用する **place-tools** および **step-init** 初期化コンテナーがマージされ、各タスクの Pod で実行されているコンテナーの数が減りました。このソリューションにより、タスクの Pod で実行されているコンテナーの数を最小限にすることにより、パイプライン実行に失敗する可能性を減らすことができます。ただし、終了メッセージの最大許容サイズの制限は削除されません。

- 今回の更新以前は、Tekton Hub Web コンソールからリソースの URL に直接アクセスしようとすると、Nginx **404** エラーが発生しました。今回の更新で、Tekton Hub Web コンソールイメージは、Tekton Hub Web コンソールから直接リソースの URL にアクセスできるように修正されました。

- 今回の更新以前は、namespace ごとにリソースプルーナージョブがリソースのプルーニング用に別個のコンテナーを作成していました。今回の更新により、リソースプルーナージョブはすべての namespace のコマンドを 1 つのコンテナーのループとして実行するようになりました。

4.1.6.6. Red Hat OpenShift Pipelines General Availability 1.7.2 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.7.2 が OpenShift Container Platform 4.9、4.10、およびそれ以降のバージョンで利用可能になりました。

4.1.6.6.1. 既知の問題

- **openshift-pipelines** namespace の Tekton Chains の **chains-config** 設定マップは、Red Hat OpenShift Pipelines Operator のアップグレード後に自動的にデフォルト値にリセットされます。現在、この問題に対する回避策はありません。

4.1.6.6.2. 修正された問題

- 今回の更新以前は、最初の引数として **init** を使用し、その後に 2 つまたはそれ以上の引数を指定した場合、Pipeline 1.7.1 のタスクは失敗していました。今回の更新により、フラグが正しく解析され、タスクが正常に実行されるようになりました。

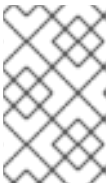
- 今回の更新以前は、無効なロールバインディングにより、OpenShift Container Platform 4.9 および 4.10 への Red Hat OpenShift Pipelines Operator のインストールは、以下のエラーメッセージと共に失敗していました。

```
error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
```

```
RoleBinding.rbac.authorization.k8s.io "openshift-operators-prometheus-k8s-read-binding" is
invalid: roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io",
Kind:"Role", Name:"openshift-operator-read"}: cannot change roleRef
```

今回の更新により、Red Hat OpenShift Pipelines Operator は個別のロールバインディング namespace でインストールし、他の Operator のインストールとの競合を回避するようになりました。

- 今回の更新以前は、Operator をアップグレードすると、Tekton Chains の **signing-secrets** シークレットキーがデフォルト値にリセットされていました。今回の更新により、カスタムシークレットキーは Operator のアップグレード後も永続するようになりました。



注記

Red Hat OpenShift Pipelines 1.7.2 へのアップグレードにより、キーがリセットされます。ただし、それ以降のリリースにアップグレードすると、キーは永続化される予定です。

- 今回の更新以前は、すべての S2I ビルドタスクが以下の様なエラーメッセージと共に失敗していました。

```
Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
operation not permitted
time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
/proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
```

今回の更新により、**pipelines-scc** セキュリティーコンテキスト制約 (SCC) は、 **Buildah** および **S2I** クラスタータスクに必要な **SETFCAP** 機能と互換性が確保されています。その結果、**Buildah** および **S2I** ビルドタスクを正常に実行できます。

さまざまな言語やフレームワークで書かれたアプリケーションに対して **Buildah** クラスタータスクおよび **S2I** ビルドタスクを正常に実行するには、**build** や **push** などの適切な **steps** オブジェクトに以下のスニペットを追加します。

```
securityContext:
  capabilities:
    add: ["SETFCAP"]
```

4.1.6.7. Red Hat OpenShift Pipelines General Availability 1.7.3 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.7.3 が OpenShift Container Platform 4.9、4.10、および 4.11 で利用可能になりました。

4.1.6.7.1. 修正された問題

- 今回の更新前は、いずれかの namespace が **Terminating** 状態の場合、RBAC リソースの作成時に Operator が失敗していました。今回の更新により、Operator は **Terminating** 状態の namespace を無視し、RBAC リソースを作成します。

- 以前は、Red Hat OpenShift Pipelines Operator をアップグレードすると **pipeline** サービスアカウントが再作成され、サービスアカウントにリンクされたシークレットが失われていました。今回の更新でこの問題が修正されています。アップグレード中に、Operator は **pipeline**

サービスアカウントを再作成しなくなりました。その結果、**pipeline** サービスアカウントにアタッチされたシークレットはアップグレード後も保持され、リソース (タスクとパイプライン) は引き続き正しく機能します。

4.1.7. Red Hat OpenShift Pipelines General Availability (GA) 1.6 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.6 が OpenShift Container Platform 4.9 で利用可能になりました。

4.1.7.1. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.6 の主な新機能について説明します。

- 今回の更新により **--output <string>** を使用して、YAML または JSON 形式の文字列を返すようにパイプラインまたはタスクの **start** コマンドを設定できるようになりました。ここでは、**<string>** は **yaml** または **json** に置き換えます。**--output** オプションを指定しないと、**start** コマンドは人間による解読はしやすくなりますが、他のプログラムによる解析が難しいメッセージを返します。継続的インテグレーション (CI) 環境では、YAML または JSON 形式の文字列を返す機能は便利です。たとえば、リソースの作成後に **yq** または **jq** を使用して、リソースに関する YAML または JSON 形式のメッセージを解析し、**showlog** オプションを使用せずにそのリソースが終了するまで待機します。

- 今回の更新により、Podman の **auth.json** 認証ファイルを使用してレジストリーに対して認証できるようになりました。たとえば、**tkn bundle push** を使用して、Docker CLI ではなく Podman を使用してリモートレジストリーにプッシュできます。

- 今回の更新により、**tkn [taskrun | pipelinerun] delete --all** コマンドを使用すると、新規の **--keep-since <minutes>** オプションを使用して、指定した期間よりも後の実行を保持できます。たとえば、5 分未満の実行を維持するには、**tkn [taskrun | pipelinerun] delete -all --keep-since 5** を入力します。

- 今回の更新により、タスク実行またはパイプライン実行を削除する際に、**--parent-resource** と **--keep-since** オプションを同時に使用できるようになりました。たとえば、**tkn pipelinerun delete --pipeline pipelinename --keep-since 5** コマンドは、親リソースの名前が **pipelinename** で、その経過時間が 5 分以下であるパイプラインの実行を保持します。**tkn tr delete -t <taskname> --keep-since 5** および **tkn tr delete --clustertask <taskname> --keep-since 5** コマンドはタスク実行と同様に機能します。

- 今回の更新により、**v1beta1** リソースと連携するトリガーリソースのサポートが追加されました。

- 今回の更新により、**ignore-running** オプションが **tkn pipelinerun delete** および **tkn taskrun delete** コマンドに追加されています。

- 今回の更新により、**create** サブコマンドが **tkn task** と **tkn clustertask** コマンドに追加されました。

- 今回の更新により、**tkn pipelinerun delete --all** コマンドを使用すると、新規の **--label <string>** オプションを使用して、ラベルでパイプライン実行をフィルターできるようになりました。オプションで、**--label** オプションに **=** と **==** を等価演算子として、または **!=** を不等価演算子として指定して使用できます。たとえば、**tkn pipelinerun delete --all --label asdf** および **tkn pipelinerun delete --all --label==asdf** コマンドはどちらも、**asdf** ラベルが割り当てられたすべてのパイプライン実行を削除します。

- 今回の更新では、設定マップからインストールされた Tekton コンポーネントのバージョンを取得するか、設定マップがない場合はデプロイメントコントローラーから取得できるようになりました。

- 今回の更新では、機能フラグを設定し、デフォルト値をそれぞれ設定するために **feature-flags** と **config-defaults** 設定マップをサポートするようになりました。

- 今回の更新では、新しいメトリクス **eventlistener_event_count** が追加され、**EventListener** リソースが受信するイベントをカウントできるようになりました。

- 今回の更新では、**v1beta1** Go API タイプが追加されました。今回の更新では、トリガーが **v1beta1** API バージョンをサポートするようになりました。
  現在のリリースでは、**v1alpha1** 機能が非推奨となり、今後のリリースで削除されます。代わりに **v1beta1** 機能の使用を開始します。

- 現在のリリースでは、リソースの自動実行がデフォルトで有効になっています。さらに、以下の新規アノテーションを使用して、namespace ごとにタスク実行およびパイプライン実行を自動実行するように設定できます。

  - **operator.tekton.dev/prune.schedule**: このアノテーションの値が **TektonConfig** カスタムリソース定義で指定された値と異なる場合には、その namespace に新規の cron ジョブが作成されます。

  - **operator.tekton.dev/prune.skip**: **true** に設定されている場合、設定先の namespace はプルーニングされません。

  - **operator.tekton.dev/prune.resources**: このアノテーションではリソースのコンマ区切りのリストを使用できます。パイプライン実行などの単一リソースをプルーニングするには、このアノテーションを **pipelinerun** に設定します。task run や pipeline run などの複数のリソースをプルーニングするには、このアノテーションを **"taskrun, pipelinerun"** に設定します。

  - **operator.tekton.dev/prune.keep**: このアノテーションを使用して、プルーニングなしでリソースを保持します。

  - **operator.tekton.dev/prune.keep-since**: このアノテーションを使用して、経過時間をもとにリソースを保持します。このアノテーションの値は、リソースの経過時間 (分単位) と等しくなければなりません。たとえば、6 日以上前に作成されたリソースを保持するには、**keep-since** を **7200** に設定します。

    注記

    keep および **keep-since** アノテーションは同時に使用できません。リソースには、どちらか 1 つだけを使用する必要があります。

  - **operator.tekton.dev/prune.strategy**: このアノテーションの値を **keep** または **keep-since** のいずれかに設定します。

- 管理者はクラスター全体に対する **pipeline** サービスアカウントの作成を無効にし、紐付けされた SCC (**anyuid** と非常に似ている) の悪用による権限昇格を防ぎます。

- **TektonConfig** カスタムリソース (CR) および、**TektonPipeline** と **TektonTriggers** などの個々のコンポーネントの CR を使用して、機能フラグおよびコンポーネントを設定できるようになりました。この詳細レベルは、個々のコンポーネントの Tekton OCI バンドルなどのアルファ機能のカスタマイズおよびテストに役立ちます。

- **PipelineRun** リソースのオプションの **Timeouts** フィールドを設定できるようになりました。たとえば、パイプライン実行、各タスク実行、および **finally** タスクに個別にタイムアウトを設定できます。

- **TaskRun** リソースで生成される Pod を使用して、Pod の **activeDeadlineSeconds** フィールドが設定されるようになりました。これにより、OpenShift はこの値を終了として考慮でき、Pod に具体的にスコープを指定した **ResourceQuota** オブジェクトを使用できます。

- configmaps を使用して、タスク実行、パイプライン実行、タスク、およびパイプラインのメトリックタグまたはラベルタイプを削除できます。さらに、ヒストグラム、ゲージ、最終値など、測定期間に、さまざまな種類のメトリックを設定できます。

- Tekton は **Min**、**Max**、**Default** および **DefaultRequest** フィールドを考慮して **LimitRange** オブジェクトを完全にサポートするため、一貫性をもたせて Pod への要求および制限を定義できます。

- 以下のアルファ機能が導入されました。

  - パイプライン実行は、以前の動作のように、すべてのタスク実行を直接停止するのではなく、**finally** タスクの実行後に停止できるようになりました。今回の更新により、以下の **spec.status** 値が追加されました。

    - **StoppedRunFinal** は、完了後、現在実行中のタスクを停止し、**finally** タスクを実行します。

    - **CancelledRun** は、実行中のタスクをすぐにキャンセルしてから、**finally** タスクを実行します。

    - **Cancelled** は、**PipelineRunCancelled** ステータスで提供される以前の動作を保持します。

      注記

      非推奨となった **PipelineRunCancelled** ステータスは **v1** バージョンで削除され、**Cancelled** ステータスに置き換えられます。

  - **oc debug** コマンドを使用して、タスク実行をデバッグモードに配置できるようになりました。これにより、実行を一時停止し、Pod で特定の手順を検査できるようになりました。

  - ステップの **onError** フィールドを **continue** に設定すると、ステップの終了コードが記録され、後続のステップに渡されます。ただし、タスク実行は失敗しないので、タスクの残りのステップの実行は継続されます。既存の動作を維持するには、**onError** フィールドの値を **stopAndFail** に設定します。

  - タスクは、実際に使用されているよりも多くのパラメーターを受け入れるようになりました。アルファ機能フラグを有効にすると、パラメーターは暗黙的にインライン仕様に伝播できます。たとえば、インラインのタスクは、タスクの各パラメーターを明示的に定義せずに、親パイプライン実行のパラメーターにアクセスできます。

  - アルファ機能のフラグを有効にすると、**when** 式の条件が、直接関連付けられたタスクにのみ適用され、タスクに依存することはありません。**When** 式を関連タスクとその依存に適用するには、式を依存タスクごとに個別に関連付ける必要があります。今後、これが Tekton の新規 API バージョンの **When** 式のデフォルト動作になることに注意してください。今回の更新が優先され、既存のデフォルト動作は非推奨になりました。

- 現在のリリースでは、**nodeSelector** および **tolerations** の値を **TektonConfig カスタムリソース** (CR) に指定することで、ノードの選択を設定できます。Operator はこれらの値を、作成するすべてのデプロイメントに追加します。
  - Operator のコントローラーおよび Webhook デプロイメントのノード選択を設定するには、Operator のインストール後に **Subscription** CR の仕様で **config.nodeSelector** および **config.tolerations** フィールドを編集します。
  - OpenShift Pipelines の残りのコントロールプレーン Pod をインフラストラクチャーノードにデプロイするには、**nodeSelector** および **tolerations** フィールドで **TektonConfig** CR を更新します。その後、変更は Operator で作成されるすべての Pod に適用されます。

4.1.7.2. 非推奨の機能

- CLI 0.21.0 では、**clustertask**、**task**、**taskrun**、**pipeline**、および **pipelinerun** コマンドに対するすべての **v1alpha1** リソースのサポートが非推奨になりました。クラスターローダーが非推奨になり、今後のリリースで削除されます。
- Tekton Triggers v0.16.0 では、重複する **status** ラベルは **EventListener** リソースのメトリックから削除されます。

重要

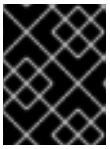
重大な変更:**status** ラベルは **eventlistener_http_duration_seconds_*** メトリックから削除されました。**status** ラベルに基づくクエリーを削除します。

- 現在のリリースでは、**v1alpha1** 機能が非推奨となり、今後のリリースで削除されます。代わりに、今回の更新では、**v1beta1** Go API タイプの使用を開始できるようになりました。トリガーが **v1beta1** API バージョンをサポートするようになりました。
- 現在のリリースでは、**EventListener** リソースはトリガーの終了処理前に応答を送信します。

重要

重大な変更: 今回の変更により、**EventListener** リソースがリソースの作成時に **201 Created** ステータスコードに応答しなくなります。代わりに **202 Accepted** 応答コードで応答します。

- 今回のリリースで、**podTemplate** フィールドが **EventListener** リソースから削除されます。



重要

重大な変更: #1100 の一部として非推奨となった **podTemplate** フィールドが削除されました。

- 今回のリリースで、非推奨の **replicas** フィールドが **EventListener** リソースの仕様から削除されます。

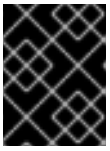
重要

重大な変更: 非推奨の **replicas** フィールドが削除されました。

- Red Hat OpenShift Pipelines 1.6 では、**HOME="/tekton/home"** および **workingDir="/workspace"** の値が **Step** オブジェクトの仕様から削除されます。
  代わりに、Red Hat OpenShift Pipelines は、**Step** オブジェクトを実行するコンテナーで定義される値に **HOME** および **workingDir** を設定します。これらの値は、**Step** オブジェクトの仕様で上書きできます。

  以前の動作を使用するには、**TektonConfig** CR の **disable-working-directory-overwrite** フィールドおよび **disable-home-env-overwrite** フィールドを **false** に変更します。

```
apiVersion: operator.tekton.dev/v1alpha1
 kind: TektonConfig
 metadata:
   name: config
 spec:
   pipeline:
     disable-working-directory-overwrite: false
     disable-home-env-overwrite: false
 ...
```



重要

TektonConfig CR の **disable-working-directory-overwrite** と **disable-home-env-overwrite** 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。

4.1.7.3. 既知の問題

- Maven および Jib Maven クラスタータスクを実行する場合には、デフォルトのコンテナーイメージは Intel(x86) アーキテクチャーでのみサポートされます。したがって、IBM Power Systems(ppc64le)、IBM Z、および LinuxONE(s390x) クラスターではタスクに失敗します。回避策として、**MAVEN_IMAGE** パラメーターの値を **maven:3.6.3-adoptopenjdk-11** に設定すると、カスタムイメージを指定できます。

- IBM Power Systems、IBM Z、および LinuxONE では、**s2i-dotnet** クラスタータスクはサポートされません。

- **tkn hub** を使用して IBM Power Systems(ppc64le)、IBM Z、および LinuxONE(s390x) の Tekton Catalog をもとにタスクをインストールする前に、タスクがこれらのプラットフォームで実行できるかどうかを確認します。**ppc64le** および **s390x** がタスク情報の Platforms セクションにリスト表示されているかどうかを確認するには、**tkn hub info task <name>** コマンドを実行します。

- **nodejs:14-ubi8-minimal** イメージストリームを使用すると、以下のエラーが生成されるため、使用できません。

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

4.1.7.4. 修正された問題

- IBM Power Systems、IBM Z、および LinuxONE では、**tkn hub** コマンドはサポート対象外になりました。
- この更新以前は、ユーザーが **tkn** コマンドの実行後にターミナルを利用できず、**再試行** が指定された場合でもパイプライン実行が行われていました。タスク実行またはパイプライン実行のタイムアウトの指定には影響がありません。今回の更新で問題が修正され、コマンド実行後にターミナルが利用できるようになります。
- 今回の更新以前は、**tkn pipelinerun delete --all** を実行すると、すべてのリソースが削除されました。今回の更新で、実行中の状態のリソースが削除されなくなりました。
- 今回の更新以前は、**tkn version --component=<component>** コマンドを使用しても、コンポーネントのバージョンが返されませんでした。今回の更新でこの問題が修正され、このコマンドを使用すると、コンポーネントのバージョンを返すようになりました。
- 今回の更新以前は、**tkn pr logs** コマンドを使用すると、パイプラインの出力ログでタスクの順番が間違って表示されていました。今回の更新で問題は解決され、完了した **PipelineRun** のログで、**TaskRun** 実行順序を適切に表示するようになりました。
- 今回の更新以前は、実行中のパイプラインの仕様を編集すると、パイプライン実行が完了時に停止できなくなる可能性がありました。今回の更新では、定義を 1 度だけフェッチし、検証用にステータスに保存されている仕様を使用して問題を修正しています。今回の変更により、**PipelineRun** または **TaskRun** が実行中の **Pipeline** または **Task** を参照する場合に競合状態に陥る確率が削減されます。
- **when** 式値に、**[$(params.arrayParam[*])]** などの配列パラメーター参照を指定できるようになりました。

4.1.7.5. Red Hat OpenShift Pipelines General Availability 1.6.1 のリリースノート

4.1.7.5.1. 既知の問題

- 古いバージョンから Red Hat OpenShift Pipelines 1.6.1 にアップグレードした後に、Pipeline は、Tekton リソース (タスクおよびパイプライン) で操作 (作成/削除/適用) を実行できない一貫性のない状態になる可能性があります。たとえば、リソースの削除中に、以下のエラーが発生する可能性があります。

  > Error from server (InternalError): Internal error occurred: failed calling webhook "validation.webhook.pipeline.tekton.dev": Post "https://tekton-pipelines-webhook.openshift-pipelines.svc:443/resource-validation?timeout=10s": service "tekton-pipelines-webhook" not found.

4.1.7.5.2. 修正された問題

- Red Hat OpenShift Pipelines によって設定される **SSL_CERT_DIR** 環境変数 (**/tekton-custom-certs**) は、以下のデフォルトのシステムディレクトリーを証明書ファイルで上書きしません。
  - **/etc/pki/tls/certs**
  - **/etc/ssl/certs**
  - **/system/etc/security/cacerts**
- Horizontal Pod Autoscaler は、Red Hat OpenShift Pipelines Operator によって制御されるデプロイメントのレプリカ数を管理できます。このリリース以降、エンドユーザーまたはクラスター上のエージェントによってカウントが変更された場合、Red Hat OpenShift Pipelines

Operator はそれによって管理されるデプロイメントのレプリカカウントをリセットしません。ただし、Red Hat OpenShift Pipelines Operator のアップグレード時にレプリカはリセットされます。

- **tkn** CLI を提供する Pod は、ノードセレクターおよび **TektonConfig** カスタムリソースで指定される容認制限に基づいて、ノードにスケジュールされるようになりました。

4.1.7.6. Red Hat OpenShift Pipelines General Availability 1.6.2 のリリースノート

4.1.7.6.1. 既知の問題

- 新規プロジェクトの作成時に、**pipeline** サービスアカウントの作成が遅延し、既存のクラスタータスクおよびパイプラインテンプレートの削除に 10 分以上かかります。

4.1.7.6.2. 修正された問題

- 今回の更新以前は、古いバージョンから Red Hat OpenShift Pipelines 1.6.1 にアップグレードした後に、Tekton インストーラーセットの複数のインスタンスがパイプライン用に作成されました。今回の更新では、Operator により、アップグレード後に **TektonInstallerSet** の各タイプのインスタンスが 1 つだけ存在するようになりました。

- 今回の更新以前は、Operator のすべてのリコンサイラーはコンポーネントバージョンを使用して、古いバージョンから Red Hat OpenShift Pipelines 1.6.1 へのアップグレード時にリソース再作成を決定していました。その結果、アップグレード時にコンポーネントのバージョンが変更されなかったリソースは再作成されませんでした。今回の更新により、Operator はコンポーネントのバージョンではなく Operator バージョンを使用して、アップグレード時にリソースの再作成を決定するようになりました。

- この更新の前は、アップグレード後にパイプライン Webhook サービスがクラスターにありませんでした。これは、設定マップのアップグレードのデッドロックが原因でした。今回の更新により、設定マップがクラスターにない場合に Webhook 検証を無効にするメカニズムが追加されました。その結果、パイプライン Webhook サービスはアップグレード後もクラスターで永続化します。

- 今回の更新以前は、namespace への設定変更後に自動プルーニングの cron ジョブは再作成されていました。今回の更新により、namespace に関連するアノテーションが変更された場合のみ、自動プルーニングの Cron ジョブは再作成されるようになりました。

- Tekton Pipelines のアップストリームバージョンは **v0.28.3** に改訂され、以下の修正が加えられました。

  - **PipelineRun** または **TaskRun** オブジェクトを修正し、ラベルまたはアノテーションの伝搬を許可します。

  - 暗黙的なパラメーターの場合:

    - **PipelineSpec** パラメーターを **TaskRefs** オブジェクトに適用しないでください。

    - **Pipeline** オブジェクトの暗黙的なパラメーター動作を無効にします。

4.1.7.7. Red Hat OpenShift Pipelines General Availability 1.6.3 のリリースノート

4.1.7.7.1. 修正された問題

- 今回の更新以前は、Red Hat OpenShift Pipelines Operator は Pipeline および Trigger などのコ

ンポーネントから Pod セキュリティーポリシーをインストールしていました。ただし、コンポーネントの一部として同梱される Pod セキュリティーポリシーは、以前のリリースで非推奨となりました。今回の更新により、Operator はコンポーネントから Pod セキュリティーポリシーをインストールするのを止めました。その結果、以下のアップグレードパスが影響を受けます。

- Pipelines 1.6.1 または 1.6.2 から Pipelines 1.6.3 にアップグレードすると、Pipelines および Triggers コンポーネントからのものを含め Pod セキュリティーポリシーが削除されます。
- Pipelines 1.5.x から 1.6.3 へのアップグレードでは、コンポーネントからインストールされる Pod セキュリティーポリシーは保持されます。クラスター管理者は、それらを手動で削除できます。

注記

今後のリリースにアップグレードすると、Red Hat OpenShift Pipelines Operator は古くなったすべての Pod セキュリティーポリシーを自動的に削除します。

- 今回の更新以前は、クラスター管理者のみが OpenShift Container Platform コンソールでパイプラインメトリックにアクセスできていました。今回の更新により、他のクラスターロールを持つユーザーもパイプラインメトリックにアクセスできるようになりました。
- 今回の更新以前は、Pipelines Operator でのロールベースアクセス制御 (RBAC) の問題により、コンポーネントのアップグレードまたはインストールに問題が生じていました。今回の更新により、各種の Red Hat OpenShift Pipelines コンポーネントをインストールする際の信頼性および一貫性が向上しました。
- 今回の更新以前は、**TektonConfig** CR で **clusterTasks** および **pipelineTemplates** フィールドを **false** に設定すると、クラスタータスクおよびパイプラインテンプレートの削除が遅くなりました。この更新により、クラスタータスクやパイプラインテンプレートなどの Tekton リソースのライフサイクル管理の速度が改善されました。

4.1.7.8. Red Hat OpenShift Pipelines General Availability (GA) 1.6.4 のリリースノート

4.1.7.8.1. 既知の問題

- Red Hat OpenShift Pipelines 1.5.2 から 1.6.4 にアップグレードした後、イベントリスナールートにアクセスすると **503** エラーが返されます。
  回避策: YAML ファイルで、イベントリスナーのルートのターゲットポートを変更します。

1. 関連する namespace のルート名を抽出します。

```
$ oc get route -n <namespace>
```

2. ルートを編集して、**targetPort** フィールドの値を変更します。

```
$ oc edit route -n <namespace> <el-route_name>
```

例: 既存のイベントリスナールート

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
```

```
  port:
    targetPort: 8000
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

例: 変更されたイベントリスナールート

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
  port:
    targetPort: http-listener
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

4.1.7.8.2. 修正された問題

- 今回の更新前は、いずれかの namespace が **Terminating** 状態の場合、RBAC リソースの作成時に Operator が失敗していました。今回の更新により、Operator は **Terminating** 状態の namespace を無視し、RBAC リソースを作成します。
- この更新の前は、関連する Tekton コントローラーのリリースバージョンを指定するアノテーションがないため、タスクの実行が失敗するか、再起動されました。今回の更新により、適切な注釈の組み込みが自動化され、タスクは失敗や再起動なしで実行されます。

4.1.8. Red Hat OpenShift Pipelines General Availability (GA) 1.5 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.5 が OpenShift Container Platform 4.8 で利用可能になりました。

4.1.8.1. 互換性およびサポート表

現在、今回のリリースに含まれる機能には テクノロジープレビュー のものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

以下の表では、機能は以下のステータスでマークされています。

TP	テクノロジープレビュー
GA	一般公開 (GA)

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表4.2 互換性およびサポート表

機能	バージョン	サポートステータス
Pipelines	0.24	GA
CLI	0.19	GA
カタログ	0.24	GA
トリガー	0.14	TP
パイプラインリソース	-	TP

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。

4.1.8.2. 新機能

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.5 の主な新機能について説明します。

- パイプライン実行およびタスク実行は、ターゲット namespace の cron ジョブによって自動的にプルーニングされます。cron ジョブは **IMAGE_JOB_PRUNER_TKN** 環境変数の値を使用して **tkn image** の値を取得します。今回の機能拡張により、以下のフィールドが **TektonConfig** カスタムリソースに導入されるようになりました。

```
...
pruner:
  resources:
    - pipelinerun
    - taskrun
  schedule: "*/5 * * * *" # cron schedule
  keep: 2 # delete all keeping n
...
```

- OpenShift Container Platform で、Tekton Add-ons コンポーネントのインストールをカスタマイズするには、**TektonConig** カスタムリソースの新規パラメーター **clusterTasks** および **pipelinesTemplates** の値を変更します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "true"
...
```

カスタマイズは、**TektonConfig** を使用してアドオンを作成するか、Tekton Add-ons を使用して直接アドオンを作成する場合に許可されます。ただし、パラメーターが渡されない場合、コントローラーはデフォルト値でパラメーターを追加します。

注記

- アドオンが **TektonConfig** カスタムリソースを使用して作成され、**Addon** カスタムリソースでパラメーター値を変更すると、**TektonConfig** カスタムリソースの値が変更を上書きします。
- **pipelinesTemplates** パラメーターの値は、**clusterTasks** パラメーターの値が **true** の場合のみ **true** に設定できます。

- **enableMetrics** パラメーターが **TektonConfig** カスタムリソースに追加されます。これを使用して、OpenShift Container Platform の Tekton Pipeline の一部であるサービスモニターを無効にすることができます。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  pipeline:
    params:
    - name: enableMetrics
      value: "true"
...
```

- プロセスレベルでメトリックをキャプチャーする EventListener OpenCensus メトリックが追加されました。
- トリガーにはラベルセレクターが追加され、ラベルを使用してイベントリスナーのトリガーを設定できるようになりました。
- インターセプターを登録する **ClusterInterceptor** カスタムリソース定義が追加され、プラグインできる新しい **Interceptor** タイプを登録できるようになりました。さらに、以下の関連する変更が行われます。
  - トリガー仕様では、**ref** フィールドが含まれる新しい API を使用してインターセプターを設定し、クラスターインターセプターを参照できます。さらに、**params** フィールドを使用して、処理用のインターセプターに渡すパラメーターを追加することができます。
  - バンドルされたインターセプター CEL、GitHub、GitLab、および BitBucket が移行されました。新しい **ClusterInterceptor** カスタムリソース定義を使用して実装されます。
  - コアインターセプターは新しい形式に移行され、古い構文を使用して作成された新しいトリガーは自動的に新しい **ref** または **params** ベースの構文に切り替わります。
- ログの表示中にタスクまたはステップの名前の接頭辞を無効にするには、**log** コマンドに **--prefix** オプションを使用します。
- 特定のコンポーネントのバージョンを表示するには、**tkn version** コマンドで新しい **--component** フラグを使用します。

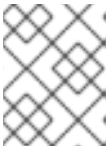
- **tkn hub check-upgrade** コマンドが追加され、他のコマンドはパイプラインのバージョンに基づいて変更されます。さらに、カタログ名は **search** コマンドの出力に表示されます。
- 任意のワークスペースのサポートは **start** コマンドに追加されます。
- プラグインが **plugins** ディレクトリーに存在しない場合は、現在のパスで検索されます。
- **tkn start [task | clustertask | pipeline]** コマンドは、対話的に開始し、デフォルトパラメーターが指定されている場合でも **params** 値の入力を求めます。対話式プロンプトを停止するには、コマンドの呼び出し時に **--use-param-defaults** フラグを渡します。以下に例を示します。

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-api \
    -p git-url=https://github.com/openshift/pipelines-vote-api.git \
    -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api \
    --use-param-defaults
```

- **version** フィールドは **tkn task describe** コマンドに追加されます。
- **TriggerTemplate**、**TriggerBinding**、**ClusterTriggerBinding**、**Eventlistener** などのリソースを自動的に選択するオプションのいずれか 1 つが存在する場合は、**describe** コマンドに追加されます。
- **tkn pr describe** コマンドでは、省略されたタスクのセクションが追加されます。
- **tkn clustertask logs** のサポートが追加されました。
- **config.yaml** からの YAML マージおよび変数は削除されます。さらに、**release.yaml** ファイルは、**kustomize** や **ytt** などのツールでより簡単に消費されるようになりました。
- ドット文字 (".") を含むリソース名のサポートが追加されました。
- **PodTemplate** 仕様の **hostAliases** 配列が、ホスト名解決の Pod レベルの上書きに追加されます。これには、**/etc/hosts** ファイルを変更します。
- タスクのアグリゲート実行ステータスにアクセスするために、変数 **$(tasks.status)** が導入されました。
- Windows のエントリーポイントバイナリービルドが追加されます。

4.1.8.3. 非推奨の機能

- **when** 式では、PascalCase で記述されたフィールドのサポートが削除されます。**when** 式は、小文字で記述されたフィールドのみをサポートします。



注記

Tekton Pipelines **v0.16** (Operator **v1.2.x**) の **when** 式のあるパイプラインを適用している場合は、これを再度適用する必要があります。

- Red Hat OpenShift Pipelines Operator を **v1.5** にアップグレードする場合、**openshift-client**

および **openshift-client-v-1-5-0** クラスタータスクには **SCRIPT** パラメーターがあります。ただし、**ARGS** パラメーターおよび **git** リソースは **openshift-client** クラスタータスクの仕様から削除されます。これは重大な変更であり、**ClusterTask** リソースの **name** フィールドに特定のバージョンのないクラスタータスクがシームレスにアップグレードされます。
パイプラインの実行が中断しないようにするには、アップグレード後に **SCRIPT** パラメーターを使用します。これは、**ARGS** パラメーターで以前に指定された値がクラスタータスクの **SCRIPT** パラメーターに移動するためです。以下に例を示します。

```
...
- name: deploy
  params:
  - name: SCRIPT
    value: oc rollout status <deployment-name>
  runAfter:
    - build
  taskRef:
    kind: ClusterTask
    name: openshift-client
...
```

- Red Hat OpenShift Pipelines Operator **v1.4** から **v1.5** にアップグレードする場合は、**TektonConfig** カスタムリソースがインストールされるプロファイル名が変更になりました。

表4.3 TektonConfig カスタムリソースのプロファイル

Pipelines 1.5 のプロファイル	Pipelines 1.4 の対応するプロファイル	インストールされた Tekton コンポーネント
すべて (**デフォルトプロファイル**)	すべて (**デフォルトプロファイル**)	Pipelines、Triggers、Add-ons
Basic	デフォルト	Pipeline、Triggers
Lite	Basic	Pipelines

注記

TektonConfig カスタムリソースの **config** インスタンスで **profile: all** を使用した場合は、リソース仕様を変更する必要はありません。

ただし、インストールされた Operator がアップグレード前に Default または Basic プロファイルのいずれかにある場合は、アップグレード後に **TektonConfig** カスタムリソースの **config** インスタンスを編集する必要があります。たとえば、アップグレードの前に設定が **profile: basic** の場合は、Pipeline 1.5 へのアップグレード後にこれが **profile: lite** であることを確認します。

- **disable-home-env-overwrite** フィールドおよび **disable-working-dir-overwrite** 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。本リリースでは、後方互換性のために、これらのフラグのデフォルト値が **true** に設定されます。

注記

次回のリリース (Red Hat OpenShift Pipelines 1.6) では、**HOME** 環境変数は自動的に **/tekton/home** に設定されず、デフォルトの作業ディレクトリーはタスク実行の **/workspace** に設定されていません。これらのデフォルトは、この手順のイメージの Dockerfile で設定されているすべての値と競合します。

- **ServiceType** フィールドおよび **podTemplate** フィールドは **EventListener** 仕様から削除されます。
- コントローラーサービスアカウントは、namespace のリスト表示および監視に対してクラスター全体のパーミッションを要求しなくなりました。
- **EventListener** リソースのステータスには、**Ready** という新規条件があります。

注記

今後、**EventListener** リソースの他のステータス条件は非推奨となり、**Ready** ステータス条件が優先されます。

- **EventListener** 応答の **eventListener** フィールドおよび **namespace** フィールドは非推奨になりました。代わりに **eventListenerUID** フィールドを使用してください。
- **replicas** フィールドは **EventListener** 仕様から非推奨になります。その代わりに、**spec.replicas** フィールドは **KubernetesResource** 仕様の **spec.resources.kubernetesResource.replicas** に移動されます。

注記

replicas フィールドは今後のリリースで削除されます。

- コアインターセプターの設定における古い方法は非推奨になりました。ただし、今後のリリースで削除されるまでこれらの作業は継続されます。代わりに、**Trigger** リソースのインターセプターが新しい **ref** および **params** ベースの構文を使用して設定されるようになりました。作成されるデフォルトの Webhook は、新規トリガーの古い構文の使用を新規構文に自動的に切り替えます。
- **ClusterRoleBinding** リソースに非推奨の **rbac.authorization.k8s.io/v1beta1** ではなく **rbac.authorization.k8s.io/v1** を使用します。
- クラスターロールでは、**serviceaccounts**、**secrets**、**configmaps**、**limitranges** などのリソースへのクラスター全体の書き込みアクセスが削除されます。さらに、**deployments**、**statefulsets**、**deployment/finalizers** などのリソースにクラスター全体のアクセスが削除されます。
- **caching.internal.knative.dev** グループの **image** カスタムリソース定義は Tekton により使用されず、本リリースで除外されます。

4.1.8.4. 既知の問題

- git-cli クラスタータスクは、alpine/git ベースイメージから構築されます。これは、**/root** がユーザーのホームディレクトリーであると想定します。ただし、これは **git-cli** クラスタータスクに明示的に設定されません。

Tekton では、特に指定がない場合は、デフォルトのホームディレクトリーはタスクのすべての手順で **/tekton/home** で上書きされます。ベースイメージの **$HOME** 環境変数を上書きすると、**git-cli** クラスタータスクが失敗します。

この問題は、今後のリリースで修正される予定です。Red Hat OpenShift Pipelines 1.5 以前のバージョンでは、**以下の回避策のいずれかを使用** して、**git-cli** クラスタータスクの失敗を防ぐことができます。

- この手順で **$HOME** 環境変数を設定します。これにより、上書きされないようにします。

  1. [オプション] Operator を使用して Red Hat OpenShift Pipeline をインストールしている場合は、**git-cli** クラスタータスクを別のタスクにクローンします。このアプローチにより、Operator はクラスタータスクに加えられた変更を上書きしないようにします。

  2. **oc edit clustertasks git-cli** コマンドを実行します。

  3. 予想される **HOME** 環境変数をステップの YAML に追加します。

```
...
steps:
  - name: git
    env:
    - name: HOME
      value: /root
    image: $(params.BASE_IMAGE)
    workingDir: $(workspaces.source.path)
...
```

警告

オペレーターがインストールした Red Hat OpenShift Pipelines の場合、**HOME** 環境変数を変更する前に **git-cli** クラスタータスクを別のタスクに複製しないと、Operator の調整中に変更が上書きされます。

- **feature-flags** 設定マップで **HOME** 環境変数の上書きを無効にします。

  1. **oc edit -n openshift-pipelines configmap feature-flags** コマンドを実行します。

  2. **disable-home-env-overwrite** フラグの値を **true** に設定します。

警告

- Operator を使用して Red Hat OpenShift Pipelines をインストールしている場合、変更は Operator の調整時に上書きされます。
- **disable-home-env-overwrite** フラグのデフォルト値を変更すると、すべてのタスクのデフォルトの動作を変更するため、他のタスクやクラスタータスクが破損する可能性があります。

  - パイプラインのデフォルトサービスアカウントが使用される場合に **HOME** 環境変数の上書きを行うため、**git-cli** クラスタータスクに別のサービスアカウントを使用します。

    1. 新規のサービスアカウントを作成します。
    2. 作成したサービスアカウントに Git シークレットをリンクします。
    3. タスクまたはパイプラインの実行中にサービスアカウントを使用します。

- IBM Power Systems、IBM Z、および LinuxONE では、**s2i-dotnet** クラスタータスクと **tkn hub** コマンドはサポートされません。
- Maven および Jib Maven クラスタータスクを実行する場合には、デフォルトのコンテナーイメージは Intel(x86) アーキテクチャーでのみサポートされます。したがって、IBM Power Systems(ppc64le)、IBM Z、および LinuxONE(s390x) クラスターではタスクに失敗します。回避策として、**MAVEN_IMAGE** パラメーターの値を **maven:3.6.3-adoptopenjdk-11** に設定すると、カスタムイメージを指定できます。

4.1.8.5. 修正された問題

- **dag** タスクの **when** 式は、他のタスクの実行ステータス (**$(tasks.<pipelineTask>.status)**) にアクセスするコンテキスト変数を指定できません。
- **PipelineRun** リソースがすぐに削除されてから再作成される状況で、**volumeClaimTemplate** PVC を削除することにより作成される競合状態を回避するのに役立つため、所有者名の代わりに所有者 UID を使用します。
- root 以外のユーザーによってトリガーされる **build-base** イメージの **pullrequest-init** に新しい Dockerfile が追加されます。
- パイプラインまたはタスクが **-f** オプションで実行され、その定義の **param** に **type** が定義されていない場合は、パイプラインまたはタスク実行が失敗する代わりに検証エラーが生成されます。
- **tkn start [task | pipeline | clustertask]** コマンドの場合は、**--workspace** フラグの説明に一貫性が保たれました。
- パラメーターを解析する際に、空の配列が発生すると、対応する対話的なヘルプが空の文字列として表示されるようになりました。

4.1.9. Red Hat OpenShift Pipelines General Availability (GA) 1.4 のリリースノート

Red Hat OpenShift Pipelines General Availability (GA) 1.4 が OpenShift Container Platform 4.7 で利用可能になりました。

注記

stable および preview Operator チャネルのほかに、Red Hat OpenShift Pipelines Operator 1.4.0 には ocp-4.6、ocp-4.5、および ocp-4.4 の非推奨チャネルが同梱されます。これらの非推奨チャネルおよびそれらのサポートは、Red Hat OpenShift Pipelines の以下のリリースで削除されます。

4.1.9.1. 互換性およびサポート表

現在、今回のリリースに含まれる機能には テクノロジープレビュー のものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

以下の表では、機能は以下のステータスでマークされています。

TP	テクノロジープレビュー
GA	一般公開 (GA)

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表4.4 互換性およびサポート表

機能	バージョン	サポートステータス
Pipelines	0.22	GA
CLI	0.17	GA
カタログ	0.22	GA
トリガー	0.12	TP
パイプラインリソース	-	TP

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。

4.1.9.2. 新機能

以下のセクションでは、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.4 の主な新機能について説明します。

- カスタムタスクには、以下の機能強化が含まれます。
  - パイプラインの結果として、カスタムタスクで生成される結果を参照できるようになりました。

- カスタムタスクはワークスペース、サービスアカウント、および Pod テンプレートを使用して、より複雑なカスタムタスクをビルドできるようになりました。

- **finally** タスクには、以下の機能強化が含まれます。
  - **when** 式は **最後** のタスクでサポートされます。これにより、効率的に保護された実行が可能になり、タスクの再利用性が向上します。
  - **finally** タスクは、同じパイプライン内のタスクの結果を使用するように設定できます。

注記

when 式および **finally** タスクのサポートは OpenShift Container Platform 4.7 Web コンソールでは利用できません。

- **dockercfg** または **dockerconfigjson** タイプの複数のシークレットのサポートがランタイム時に認証用に追加されました。

- **git-clone** タスクでスパースチェックをサポートする機能が追加されました。これにより、ローカルコピーとしてリポジトリーのサブセットのみをクローンすることができ、これはクローン作成したリポジトリーのサイズを制限するのに便利です。

- 実際に起動せずに、パイプライン実行を保留中の状態で作成できます。負荷が大きいクラスターでは、これにより、Operator はパイプライン実行の開始時間を制御することができます。

- コントローラー用に **SYSTEM_NAMESPACE** 環境変数を手動で設定していることを確認します。これは以前はデフォルトで設定されていました。

- root 以外のユーザーがパイプラインのビルドベースイメージに追加され、**git-init** がリポジトリーのクローンを root 以外のユーザーとして作成できるようになりました。

- パイプライン実行の開始前に解決されたリソース間で依存関係を検証するサポートが追加されています。パイプラインのすべての結果変数は有効でなければならず、パイプラインからのオプションのワークスペースは、パイプライン実行の開始に使用することが予想されているタスクにのみ渡すことができます。

- コントローラーおよび Webhook は root 以外のグループとして実行され、それらの必要以上の機能は削除され、よりセキュアになりました。

- **tkn pr logs** コマンドを使用して、再試行されたタスク実行のログストリームを表示できます。

- **tkn tr delete** コマンドで **--clustertask** オプションを使用して、特定のクラスタータスクに関連付けられたすべてのタスク実行を削除できます。

- **EventListener** リソースでの Knative サービスのサポートは、新規の **customResource** フィールドを導入して追加されます。

- イベントペイロードが JSON 形式を使用しない場合にエラーメッセージが表示されます。

- GitLab、BitBucket、GitHub などのソース制御インターセプターは、新規の **InterceptorRequest** または **InterceptorResponse** を使用できるようになりました。

- 新しい CEL 関数の **marshalJSON** が実装され、JSON オブジェクトまたは配列を文字列にエンコードできます。

- CEL およびソース制御コアインターセプターを提供する HTTP ハンドラーが追加されました。これは、**tekton-pipelines** namespace にデプロイされる単一の HTTP サーバーに 4 つのコアイ

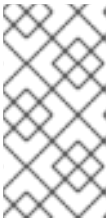
ンターセプターをパッケージ化します。**EventListener** オブジェクトは、HTTP サーバー経由でイベントをインターセプターに転送します。それぞれのインターセプターは異なるパスで利用できます。たとえば、CEL インターセプターは **/cel** パスで利用できます。

- **pipelines-scc** SCC (Security Context Constraint) は、パイプラインのデフォルト **pipeline** サービスアカウントで使用されます。この新規サービスアカウントは **anyuid** と似ていますが、OpenShift Container Platform 4.7 の SCC について YAML に定義されるように若干の違いがあります。

```
fsGroup:
  type: MustRunAs
```

4.1.9.3. 非推奨の機能

- パイプラインリソースストレージの **build-gcs** サブタイプ、および **gcs-fetcher** イメージは、サポートされていません。
- クラスタータスクの **taskRun** フィールドで、**tekton.dev/task** ラベルが削除されます。
- Webhook の場合、フィールド **admissionReviewVersions** に対応する値 **v1beta1** は削除されます。
- ビルドおよびデプロイ用の **creds-init** ヘルパーイメージが削除されます。
- トリガー仕様およびバインディングでは、**template.ref** が優先されるため、非推奨フィールドの **template.name** が削除されます。**ref** フィールドを使用するには、**eventListener** のすべての定義を更新する必要があります。



注記

template.name フィールドが利用できないため、Pipelines 1.3.x 以前のバージョンから Pipelines 1.4.0 へのアップグレードにより、イベントリスナーが破損します。このような場合には、Pipelines 1.4.1 を使用して、復元された **template.name** フィールドを利用します。

- **EventListener** カスタムリソース/オブジェクトの場合、**Resource** が優先されるために、**PodTemplate** および **ServiceType** フィールドは非推奨になりました。
- 非推奨の仕様スタイルの埋め込みバインディングは削除されています。
- **spec** フィールドは **triggerSpecBinding** から削除されています。
- イベント ID 表現は、5 文字のランダムな文字列から UUID に変更されています。

4.1.9.4. 既知の問題

- **Developer** パースペクティブでは、Pipeline メトリックおよびトリガー機能は OpenShift Container Platform 4.7.6 以降のバージョンでのみ利用できます。
- IBM Power Systems、IBM Z、および LinuxONE では、**tkn hub** コマンドはサポートされません。
- IBM Power Systems (ppc64le)、IBM Z、および LinuxONE (s390x) クラスターで Maven および Jib Maven クラスタータスクを実行する場合、**MAVEN_IMAGE** パラメーターの値を **maven:3.6.3-adoptopenjdk-11** に設定します。

- トリガーは、トリガーバインディングに以下の設定がある場合は、JSON 形式の正しくない処理によって生じるエラーを出力します。

```
params:
  - name: github_json
    value: $(body)
```

この問題を解決するには、以下を実行します。

  - トリガー v0.11.0 以降を使用している場合、**marshalJSON** 関数を使用して JSON オブジェクトまたは配列を取得し、そのオブジェクトまたは配列の JSON エンコーディングを文字列として返します。

  - 古いバージョンのトリガーを使用している場合は、以下のアノテーションをトリガーテンプレートに追加します。

```
annotations:
  triggers.tekton.dev/old-escape-quotes: "true"
```

- Pipelines 1.3.x から 1.4.x にアップグレードする際に、ルートを再作成する必要があります。

4.1.9.5. 修正された問題

- 以前のバージョンでは、**tekton.dev/task** ラベルがクラスタータスクのタスク実行から削除され、**tekton.dev/clusterTask ラベル** が導入されました。この変更により生じる問題は 、**clustertask describe** および **delete** コマンドを修正して解決されています。さらに、タスクの **lastrun** 機能は変更され、古いバージョンのパイプラインでタスクとクラスタータスクの両方のタスク実行に適用される **tekton.dev/task** ラベルの問題を修正できるになりました。

- 対話的な **tkn pipeline start pipelinename** を実行する場合、**PipelineResource** が対話的に作成されます。**tkn p start** コマンドは、リソースのステータスが **nil** ではない場合にリソースのステータスを出力します。

- 以前のバージョンでは、**tekton.dev/task=name** ラベルは、クラスタータスクから作成されるタスク実行から削除されました。今回の修正により、**--last** フラグの指定される **tkn clustertask start** コマンドが変更され、作成されたタスク実行で **tekton.dev/task=name** ラベルの有無がチェックされるようになりました。

- タスクがインラインのタスク仕様を使用する場合、対応するタスク実行は **tkn pipeline describe** コマンドの実行時にパイプラインに組み込まれ、タスク名は埋め込まれた状態で返されます。

- **tkn version** コマンドは、設定された **kubeConfiguration namespace** やクラスターへのアクセスなしに、インストールされた Tekton CLI ツールのバージョンを表示するように修正されています。

- 引数が予期せずに使用されるか、複数の引数が使用される場合、**tkn completion** コマンドでエラーが発生します。

- 以前のバージョンでは、パイプライン仕様でネスト化された **finally** タスクのあるパイプライン実行は、**v1alpha1** バージョンに変換され、**v1beta1** バージョンに戻されると、それらの **finally** タスクを失うことがあります。変換中に発生するこのエラーは修正され、潜在的データ損失を防ぐことができます。**finally** タスクがパイプライン仕様でネスト化されたパイプライン実行はシリアライズされ、アルファバージョンに保存されてデシリアライズは後に実行されるようになりました。

- 以前のバージョンでは、サービスアカウントで **secrets** フィールドに **{}** があると、Pod の生成でエラーが発生しました。空のシークレット名を持つ GET 要求がエラーがリソース名が空ではないことを示すエラーを返すため、タスク実行は **CouldntGetTask** で失敗しました。この問題は、**kubeclient** GET 要求で空のシークレット名を使用しないことで解決されています。

- **v1beta1** API バージョンのあるパイプラインは、**finally** タスクを失うことなく、**v1alpha1** バージョンと共に要求できるようになりました。返される **v1alpha1** バージョンを適用すると、リソースが **v1beta1** として保存され、**finally** セクションがその元の状態に戻ります。

- 以前のバージョンでは、コントローラーの **selfLink** フィールドが設定されていないと、Kubernetes v1.20 クラスターでエラーが発生しました。一時的な修正として、**CloudEvent** ソースフィールドは、自動設定される **selfLink** フィールドの値なしに現在のソース URI に一致する値に設定されます。

- 以前のバージョンでは、**gcr.io** などのドットの付いたシークレット名により、タスク実行の作成が失敗しました。これは、シークレット名がボリュームマウント名の一部として内部で使用されるために生じました。ボリュームマウント名は RFC1123 DNS ラベルに準拠し、名前の一部として使用されるドットを許可しません。この問題は、ドットをダッシュに置き換えることで解決し、これにより名前の読み取りが可能になりました。

- コンテキスト変数は、**finally** タスクで検証されるようになりました。

- 以前のバージョンでは、タスク実行リコンサイラーが渡され、作成した Pod の名前を含む直前のステータス更新を持たないタスク実行があると、タスク実行リコンサイラーはタスク実行に関連付けられた Pod をリスト表示しました。タスク実行リコンサイラーは、Pod を検索するために、Pod に伝播されるタスク実行のラベルを使用しました。タスク実行の実行中にこれらのラベルを変更すると、コードが既存の Pod を見つけることができませんでした。その結果、重複した Pod が作成されました。この問題は、Pod の検索時に **tekton.dev/taskRun** の Tekton で制御されるラベルのみを使用するようにタスク実行リコンサイラーを変更することで修正されています。

- 以前のバージョンでは、パイプラインがオプションのワークスペースを受け入れ、これをパイプラインタスクに渡すと、パイプライン実行リコンサイラーは、ワークスペースが提供されておらず、欠落しているワークスペースのバインディングがオプションのワークスペースについて有効な場合でも、エラーを出して停止しました。この問題は、オプションのワークスペースが指定されていない場合でも、パイプライン実行リコンサイラーがタスク実行の作成に失敗しないようにすることで修正されています。

- ステップのステータスの並び順は、ステップコンテナーの順序と一致します。

- 以前のバージョンでは、Pod で **CreateContainerConfigError** の理由が出されると、タスク実行のステータスは **unknown** に設定されました。これは、タスクおよびパイプラインが Pod がタイムアウトするまで実行されることを意味しました。この問題は、Pod で **CreateContainerConfigError** の理由が出される際にタスクを失敗 (failed) として設定できるようにタスク実行ステータスを **false** に設定することで解決されています。

- 以前のバージョンでは、パイプライン実行の完了後に、パイプラインの結果は最初の調整で解決されました。これにより解決が失敗し、パイプライン実行の **Succeeded** 状態が上書きされる可能性がありました。その結果、最終のステータス情報が失われ、パイプライン実行の状態を監視するすべてのサービスに混乱を生じさせる可能性がありました。この問題は、パイプライン実行が **Succeeded** または **True** 状態になる際に、パイプラインの結果の解決を調整の最後に移行することにで解決されました。

- 実行ステータス変数が検証されるようになりました。これにより、実行ステータスにアクセスするためのコンテキスト変数の検証中に、タスク結果が検証されることを防ぐことができます。

- 以前のバージョンでは、無効な変数を含むパイプラインの結果は、変数のリテラル式はそのままの状態でパイプライン実行に追加されます。そのため、結果が正しく設定されているかどうかを評価することは容易ではありませんでした。この問題は、失敗したタスク実行を参照するパイプライン実行結果でフィルタリングすることで解決されています。無効な変数を含むパイプラインの結果は、パイプライン実行によって出されなくなりました。

- **tkn eventlistener describe** コマンドは、テンプレートなしでクラッシュを回避できるように修正されています。また、トリガーの参照に関する情報も表示します。

- **template.name** が利用できないため、Pipelines 1.3.x 以前のバージョンから Pipelines 1.4.0 へのアップグレードにより、イベントリスナーが破損します。Pipelines 1.4.1 では、トリガーでイベントリスナーが破損しないように、**template.name** が復元されています。

- Pipelines 1.4.1 では、**ConsoleQuickStart** カスタムリソースが OpenShift Container Platform 4.7 の機能および動作に合わせて更新されました。

4.1.10. Red Hat OpenShift Pipelines テクノロジープレビュー 1.3 のリリースノート

4.1.10.1. 新機能

Red Hat OpenShift Pipelines テクノロジープレビュー (TP) 1.3 が OpenShift Container Platform 4.7 で利用可能になりました。Red Hat OpenShift Pipelines TP 1.3 が以下をサポートするように更新されています。

- Tekton Pipelines 0.19.0

- Tekton **tkn** CLI 0.15.0

- Tekton Triggers 0.10.2

- Tekton Catalog 0.19.0 をベースとするクラスタータスク

- OpenShift Container Platform 4.7 での IBM Power Systems

- OpenShift Container Platform 4.7 での IBM Z および LinuxONE

以下のセクションでは、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.3 の主な新機能について説明します。

4.1.10.1.1. Pipelines

- S2I や Buildah タスクなどのイメージをビルドするタスクが、イメージの SHA を含むビルドされたイメージの URL を生成するようになりました。

- **Condition** カスタムリソース定義 (CRD) が非推奨となっているため、カスタムタスクを参照するパイプラインタスクの条件は許可されません。

- **spec.steps[].imagePullPolicy** および **spec.sidecar[].imagePullPolicy** フィールドの **Task** CRD に変数の拡張が追加されました。

- **disable-creds-init** feature-flag を **true** に設定すると、Tekton のビルトイン認証情報メカニズムを無効にすることができます。

- 解決済みの When 式は、**PipelineRun** 設定の **Status** フィールドの **Skipped Tasks** および **Task Runs** セクションにリスト表示されるようになりました。

- **git init** コマンドが、再帰的なサブモジュールのクローンを作成できるようになりました。
- **Task** CR の作成者は、**Task** 仕様のステップのタイムアウトを指定できるようになりました。
- エントリーポイントイメージを **distroless/static:nonroot** イメージにベースとして作成し、ベースイメージに存在する **cp** コマンドを使用せずに、これを宛先にコピーするモードを許可できるようになりました。
- Git SSH シークレットの既知のホストの省略を許可しないように、設定フラグ **require-git-ssh-secret-known-hosts** を使用できるようになりました。フラグ値が **true** に設定されている場合には、Git SSH シークレットに **known_host** フィールドを含める必要があります。フラグのデフォルト値は **false** です。
- オプションのワークスペースの概念が導入されました。タスクまたはパイプラインはワークスペースオプションを宣言し、その存在に基づいて動作を条件的に変更する可能性があります。タスク実行またはパイプライン実行により、そのワークスペースが省略され、タスクまたはパイプラインの動作が変更される可能性があります。デフォルトのタスク実行ワークスペースは、省略されたオプションのワークスペースの代わりに追加されることはありません。
- Tekton の認証情報の初期化により、SSH 以外の URL で使用する SSH 認証情報が検出されるほか、Git パイプラインリソースでは SSH URL で使用する http 認証情報が検出され、Step コンテナーで警告がログに記録されるようになりました。
- タスク実行コントローラーは、Pod テンプレートで指定されたアフィニティーがアフィニティーアシスタントによって上書きされる場合に警告イベントを生成します。
- タスク実行リコンサイラーは、タスク実行が完了すると生成されるクラウドイベントのメトリックを記録するようになりました。これには再試行が含まれます。

4.1.10.1.2. Pipelines CLI

- **--no-headers flag** のサポートが、次のコマンドに追加されました: **tkn condition list**、**tkn triggerbinding list**、**tkn eventlistener list**、**tkn clustertask list**、**tkn clustertriggerbinding list**
- 併用した場合、**--last** または **--use** オプションは、**--prefix-name** および **--timeout** オプションを上書きします。
- **tkn eventlistener logs** コマンドが追加され、**EventListener** ログが表示されるようになりました。
- **tekton hub** コマンドは **tkn** CLI に統合されるようになりました。
- **--nocolour** オプションは **--no-color** に変更されました。
- **--all-namespaces** フラグは、次のコマンドに追加されました: **tkn triggertemplate list**、**tkn condition list**、**tkn triggerbinding list**、**tkn eventlistener list**

4.1.10.1.3. トリガー

- **EventListener** テンプレートでリソース情報を指定できるようになりました。
- すべてのトリガーリソースの **get** 動詞に加えて、**EventListener** サービスアカウントに **list** および **watch** 動詞が設定されることが必須になりました。これにより、**Listers** を使用して **EventListener**、 **Trigger**、**TriggerBinding**、**TriggerTemplate**、および

ClusterTriggerBinding リソースからデータを取得することができます。この機能を使用して、複数のインフォーマーを指定するのではなく **Sink** オブジェクトを作成し、API サーバーを直接呼び出すことができます。

- イミュータブルな入力イベント本体をサポートする新たな **Interceptor** インターフェイスが追加されました。インターセプターはデータまたはフィールドを新しい **extensions** フィールドに追加できるようになり、入力本体を変更できなくなったことでイミュータブルとなりました。CEL インターセプターはこの新たな **Interceptor** インターフェイスを使用します。

- **namespaceSelector** フィールドは **EventListener** リソースに追加されます。これを使用して、**EventListener** リソースがイベント処理用に **Trigger** オブジェクトを取得できる namespace を指定します。**namespaceSelector** フィールドを使用するには、**EventListener** のサービスアカウントにクラスターロールが必要です。

- トリガー **EventListener** リソースは、**eventlistener** Pod へのエンドツーエンドのセキュアな接続をサポートするようになりました。

- **"** を **\"** に置き換えることで、**TriggerTemplates** リソースのエスケープパラメーター動作が削除されました。

- Kubernetes リソースをサポートする新規 **resources** フィールドは、**EventListener** 仕様の一部として導入されます。

- ASCII 文字列の大文字と小文字へのサポートが含まれる CEL インターセプターの新機能が追加されました。

- **TriggerBinding** リソースは、トリガーの **name** および **value** フィールドを使用するか、イベントリスナーを使用して埋め込むことができます。

- **PodSecurityPolicy** 設定は、制限された環境で実行されるように更新されます。これにより、コンテナーは root 以外のユーザーとして実行する必要があります。さらに、Pod セキュリティーポリシーを使用するためのロールベースのアクセス制御は、クラスタースコープから namespace スコープに移行されます。これにより、トリガーは namespace に関連しない他の Pod セキュリティーポリシーを使用することができません。

- 埋め込みトリガーテンプレートのサポートが追加されました。**name** フィールドを使用して埋め込みテンプレートを参照するか、**spec** フィールド内にテンプレートを埋め込むことができます。

4.1.10.2. 非推奨の機能

- **PipelineResources** CRD を使用する Pipeline テンプレートは非推奨となり、今後のリリースで削除されます。

- **template.ref** フィールドが優先されるため、**template.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。

- **--check** コマンドの短縮形である **-c** が削除されました。さらに、グローバル **tkn** フラグが **version** コマンドに追加されます。

4.1.10.3. 既知の問題

- CEL オーバーレイは、受信イベント本体を変更する代わりに、フィールドを新しい最上位の **extensions** 関数に追加します。**TriggerBinding** リソースは、**$(extensions.<key>)** 構文を使用して、この新しい **extensions** 関数内の値にアクセスできます。**$(body.<overlay-key>)** の代わりに **$(extensions.<key>)** 構文を使用するようにバインディングを更新します。

- **"** を **\"** に置き換えることで、エスケープパラメーター動作が削除されました。古いエスケープパラメーターの動作を保持する必要がある場合は、**tekton.dev/old-escape-quotes: true"** アノテーションを **TriggerTemplate** 仕様に追加します。

- **TriggerBinding** リソースは、トリガーまたはイベントリスナー内の **name** および **value** フィールドを使用して組み込みことができます。ただし、単一のバインディングに **name** および **ref** フィールドの両方を指定することはできません。**ref** フィールドを使用して **TriggerBinding** リソースおよび埋め込みバインディングの **name** フィールドを参照します。

- インターセプターは、**EventListener** リソースの namespace 外で **secret** の参照を試行することはできません。シークレットを `EventListener` の namespace に含める必要があります。

- Trigger 0.9.0 以降では、本体またはヘッダーベースの **TriggerBinding** パラメーターが見つからないか、イベントペイロードで形式が正しくない場合に、エラーを表示する代わりにデフォルト値が使用されます。

- JSON アノテーションを修正するには、Tekton および Pipelines 0.16.x を使用して **WhenExpression** オブジェクトで作成されたタスクおよびパイプラインを再適用する必要があります。

- パイプラインがオプションのワークスペースを受け入れ、これをタスクに付与すると、ワークスペースが指定されていない場合はパイプライン実行が停止します。

- 非接続環境で Buildah クラスタータスクを使用するには、Dockerfile が内部イメージストリームをベースイメージとして使用していることを確認してから、これを S2I クラスタータスクと同じ方法で使用します。

4.1.10.4. 修正された問題

- CEL インターセプターによって追加された拡張機能は、イベント本体内に **Extensions** フィールドを追加して Webhook インターセプターに渡されます。

- ログリーダーのアクティビティータイムアウトは、**LogOptions** フィールドを使用して設定できるようになりました。ただし、10 秒のタイムアウトのデフォルト動作は保持されます。

- **log** コマンドは、タスク実行またはパイプライン実行が完了したときに **--follow** フラグを無視し、ライブログではなく利用可能なログを読み取ります。

- 以下の Tekton リソースへの参照:
EventListener、**TriggerBinding**、**ClusterTriggerBinding**、**Condition**、および **TriggerTemplate** は、**tkn** コマンドのすべてのユーザーに表示されるメッセージで標準化され、一貫性を保つようになりました。

- 以前は、**--use-taskrun <canceled-task-run-name>**、**--use-pipelinerun <canceled-pipeline-run-name>** または **--last** フラグを使用してキャンセルされたタスク実行またはパイプライン実行を開始した場合、新規の実行はキャンセルされました。このバグは修正されています。

- **tkn pr desc** コマンドが強化され、パイプラインが各種の状態で実行された場合に失敗しなくなりました。

- **--task** オプションで **tkn tr delete** コマンドを使用してタスク実行を削除し、クラスタータスクが同じ名前で存在する場合、クラスタータスクのタスク実行も削除されます。回避策として、**TaskRefKind** フィールドを使用して、タスク実行をフィルタリングします。

- **tkn triggertemplate describe** コマンドは、出力内の **apiVersion** 値の一部のみを表示します。たとえば、**triggers.tekton.dev/v1alpha1** ではなく、**triggers.tekton.dev** のみが表示されました。このバグは修正されています。

- 特定の条件下で Webhook はリースの取得に失敗し、正常に機能しません。このバグは修正されています。
- v0.16.3 で作成した When 式を持つパイプラインは、v0.17.1 以降で実行できるようになりました。アップグレード後に、アノテーションの最初の大文字と小文字の両方がサポートされるようになったため、以前のバージョンで作成されたパイプライン定義を再適用する必要はありません。
- デフォルトでは、**leader-election-ha** フィールドが高可用性に対して有効にされるようになりました。コントローラーフラグ **disable-ha** を **true** に設定すると、高可用性サポートが無効になります。
- 重複したクラウドイベントに関する問題が修正されています。クラウドイベントは、条件が状態、理由、またはメッセージを変更する場合にのみ送信されるようになりました。
- サービスアカウント名が **PipelineRun** または **TaskRun** 仕様にない場合、コントローラーは **config-defaults** 設定マップからサービスアカウント名を使用します。サービスアカウント名が **config-defaults** 設定マップにもない場合、コントローラーはこれを仕様で **default** に設定するようになりました。
- アフィニティーアシスタントとの互換性の検証は、同じ永続ボリューム要求 (PVC) が複数のワークスペースに使用される場合にサポートされるようになりましたが、サブパスは異なります。

4.1.11. Red Hat OpenShift Pipelines テクノロジープレビュー 1.2 のリリースノート

4.1.11.1. 新機能

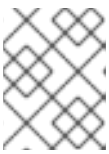
Red Hat OpenShift Pipelines テクノロジープレビュー (TP) 1.2 が OpenShift Container Platform 4.6 で利用可能になりました。Red Hat OpenShift Pipelines TP 1.2 が以下をサポートするように更新されています。

- Tekton Pipelines 0.16.3
- Tekton **tkn** CLI 0.13.1
- Tekton Triggers 0.8.1
- Tekton Catalog 0.16 をベースとするクラスタータスク
- OpenShift Container Platform 4.6 での IBM Power Systems
- OpenShift Container Platform 4.6 での IBM Z および LinuxONE

以下では、修正および安定性の面での改善点に加え、OpenShift Pipelines 1.2 の主な新機能について説明します。

4.1.11.1.1. Pipelines

- Red Hat OpenShift Pipelines のリリースでは、非接続インストールのサポートが追加されました。



注記

制限された環境でのインストールは現時点で、IBM Power Systems、IBM Z、および LinuxONE ではサポートされていません。

- **conditions** リソースの代わりに **when** フィールドを使用して、特定の条件が満たされる場合にのみタスクを実行できるようになりました。**WhenExpression** の主なコンポーネントは **Input**、 **Operator**、および **Values** です。すべての When 式が **True** に評価されると、タスクが実行されます。When 式のいずれかが **False** に評価されると、タスクはスキップされます。

- ステップのステータスは、タスクの実行がキャンセルまたはタイムアウトすると更新されるようになりました。

- **git-init** が使用するベースイメージをビルドするために、Git Large File Storage (LFS) のサポートが利用できるようになりました。

- **taskSpec** フィールドを使用して、タスクがパイプラインに組み込まれる際に、ラベルやアノテーションなどのメタデータを指定できるようになりました。

- クラウドイベントがパイプラインの実行でサポートされるようになりました。**backoff** を使用した再試行が、クラウドイベントパイプラインリソースによって送信されるクラウドイベントに対して有効になりました。

- **Task** リソースが宣言するものの、**TaskRun** リソースが明示的に指定しないワークスペースのデフォルトの **Workspace** 設定を設定できるようになりました。

- サポートは、**PipelineRun** namespace および **TaskRun** namespace の namespace 変数の補間に利用できます。

- **TaskRun** オブジェクトの検証が追加され、**TaskRun** リソースが Affinity Assistant に関連付けられる際に複数の永続ボリューム要求 (PVC) ワークスペースが使用されていないことを確認するようになりました。複数の永続ボリューム要求 (PVC) ワークスペースが使用されていると、タスクの実行は **TaskRunValidationFailed** の状態で失敗します。デフォルトで、Affinity Assistant は Red Hat OpenShift Pipelines で無効にされているため、これを使用できるように有効にする必要があります。

4.1.11.1.2. Pipelines CLI

- **tkn task describe**、**tkn taskrun describe**、**tkn clustertask describe**、**tkn pipeline describe**、および **tkn pipelinerun describe** コマンドが以下を実行するようになりました。
  - **Task**、**TaskRun**、**ClusterTask**、**Pipeline** および **PipelineRun** リソースのいずれかが 1 つしかない場合、それぞれを自動的に選択します。
  - 出力に **Task**、**TaskRun**、**ClusterTask**、**Pipeline** および **PipelineRun** リソースの結果をそれぞれ表示します。
  - 出力に **Task**、**TaskRun**、**ClusterTask**、**Pipeline** および **PipelineRun** リソースで宣言されたワークスペースをそれぞれ表示します。

- **tkn clustertask start** コマンドに **--prefix-name** オプションを指定して、タスク実行の名前に接頭辞を指定でき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 **tkn clustertask start** コマンドに提供されるようになりました。

- **TaskRun** および **PipelineRun** オブジェクトのローカルまたはリモートファイル定義を使用して、パイプラインでサポートされる **PodTemplate** プロパティーを指定できるようになりました。

- **--use-params-defaults** オプションを **tkn clustertask start** コマンドに指定して、**ClusterTask** 設定に設定したデフォルト値を使用して、タスク実行を作成できるようになりました。

- **tkn pipeline start** コマンドの **--use-param-defaults** フラグで、デフォルトの値が一部のパラメーターに指定されていない場合に対話モードをプロンプトで表示するようになりました。

4.1.11.1.3. トリガー

- YAML 文字列を文字列のマップに解析するために、**parseYAML** という名前の Common Expression Language (CEL) 関数が追加されました。
- 式を評価する際や、評価環境を作成するためにフック本体を解析する際に、CEL 式の解析を行うエラーメッセージの詳細度が上がりました。
- ブール値とマップが CEL オーバーレイメカニズムで式の値として使用されている場合に、それらをマーシャリングするためのサポートが利用できるようになりました。
- 以下のフィールドが **EventListener** オブジェクトに追加されました。
  - **replicas** フィールドは、YAML ファイルのレプリカ数を指定して、イベントリスナーが複数の Pod を実行できるようにします。
  - **NodeSelector** フィールドでは、**EventListener** オブジェクトがイベントリスナー Pod を特定のノードにスケジュールできるようにします。
- Webhook インターセプターは **EventListener-Request-URL** ヘッダーを解析し、イベントリスナーによって処理される元のリクエスト URL からパラメーターを抽出できるようになりました。
- イベントリスナーからのアノテーションがデプロイメント、サービス、およびその他の Pod に伝播できるようになりました。サービスまたはデプロイメントのカスタムアノテーションは上書きされるため、イベントリスナーアノテーションに追加して伝播できるようにする必要があります。
- **EventListener** 仕様のレプリカの適切な検証が、ユーザーが **spec.replicas** 値を **negative** または **zero** として指定する場合に利用できるようになりました。
- **TriggerCRD** オブジェクトを、**TriggerRef** フィールドを使用して参照として **EventListener** 仕様内に指定し、**TriggerCRD** オブジェクトを別個に作成してから、これを **EventListener** 仕様内でバインドできるようになりました。
- **TriggerCRD** オブジェクトの検証およびデフォルト値が利用可能になりした。

4.1.11.2. 非推奨の機能

- **$(params)** パラメーターは **triggertemplate** リソースから削除され、**$(tt.params)** に置き換えられ、これにより **resourcetemplate** と **triggertemplate** パラメーター間の混乱が生じなくなります。
- オプションの **EventListenerTrigger** ベースの認証レベルの **ServiceAccount** 参照が **ServiceAccountName** 文字列へのオブジェクト参照から変更されました。これにより、**ServiceAccount** 参照が **EventListenerTrigger** オブジェクトと同じ namespace に置かれるようになりました。
- **Conditions** カスタムリソース定義 (CRD) は非推奨となり、代わりに **WhenExpressions** CRD が使用されます。
- **PipelineRun.Spec.ServiceAccountNames** オブジェクトは非推奨となり、**PipelineRun.Spec.TaskRunSpec[].ServiceAccountName** オブジェクトによって置き換えられます。

4.1.11.3. 既知の問題

- Red Hat OpenShift Pipelines のリリースでは、非接続インストールのサポートが追加されました。ただし、クラスタータスクで使用される一部のイメージは、非接続クラスターで動作するようにミラーリングする必要があります。

- **openshift** namespace のパイプラインは、Red Hat OpenShift Pipelines Operator のアンインストール後に削除されません。**oc delete pipelines -n openshift --all** コマンドを使用してパイプラインを削除します。

- Red Hat OpenShift Pipelines Operator をアンインストールしても、イベントリスナーは削除されません。
  回避策として、**EventListener** および **Pod** CRD を削除するには、以下を実行します。

  1. **EventListener** オブジェクトを **foregroundDeletion** ファイナライザーで編集します。

     ```
     $ oc patch el/<eventlistener_name> -p '{"metadata":{"finalizers":["foregroundDeletion"]}}'
     --type=merge
     ```

     以下に例を示します。

     ```
     $ oc patch el/github-listener-interceptor -p '{"metadata":{"finalizers":
     ["foregroundDeletion"]}}' --type=merge
     ```

  2. **EventListener** CRD を削除します。

     ```
     $ oc patch crd/eventlisteners.triggers.tekton.dev -p '{"metadata":{"finalizers":[]}}' --
     type=merge
     ```

- IBM Power Systems (ppc64le) または IBM Z (s390x) クラスターでコマンド仕様なしにマルチアーキテクチャーコンテナーイメージタスクを実行すると、**TaskRun** リソースは以下のエラーを出して失敗します。

  ```
  Error executing command: fork/exec /bin/bash: exec format error
  ```

  回避策として、アーキテクチャー固有のコンテナーイメージを使用するか、正しいアーキテクチャーを参照する sha256 ダイジェストを指定します。sha256 ダイジェストを取得するには、以下を実行します。

  ```
  $ skopeo inspect --raw <image_name>| jq '.manifests[] | select(.platform.architecture == "
  <architecture>") | .digest'
  ```

4.1.11.4. 修正された問題

- CEL フィルター、Webhook バリデーターのオーバーレイ、およびインターセプターの式を確認するための簡単な構文検証が追加されました。

- Trigger は、基礎となるデプロイメントおよびサービスオブジェクトに設定されたアノテーションを上書きしなくなりました。

- 以前のバージョンでは、イベントリスナーはイベントの受け入れを停止しました。今回の修正により、この問題を解決するために **EventListener** シンクの 120 秒のアイドルタイムアウトが追加されました。

- 以前のバージョンでは、**Failed(Canceled)** 状態でパイプラインの実行を取り消すと、成功のメッセージが表示されました。これは、代わりにエラーが表示されるように修正されました。

- **tkn eventlistener list** コマンドがリスト表示されたイベントリスナーのステータスを提供するようになり、利用可能なイベントリスナーを簡単に特定できるようになりました。

- トリガーがインストールされていない場合や、リソースが見つからない場合に、**triggers list** および **triggers describe** コマンドについて一貫性のあるエラーメッセージが表示されるようになりました。

- 以前のバージョンでは、多くのアイドル接続がクラウドイベントの配信時に増大しました。この問題を修正するために、**DisableKeepAlives: true** パラメーターが **cloudeventclient** 設定に追加されました。新規の接続がすべてのクラウドイベントに設定されます。

- 以前のバージョンでは、特定のタイプの認証情報が指定されていない場合であっても、**creds-init** コードが空のファイルをディスクに書き込みました。今回の修正により、**creds-init** コードが変更され、正しくアノテーションが付けられたシークレットから実際にマウントされた認証情報のみのファイルを書き込むようになりました。

4.1.12. Red Hat OpenShift Pipelines テクノロジープレビュー 1.1 のリリースノート

4.1.12.1. 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 をベースとするクラスタータスク

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

4.1.12.1.1. Pipelines

- ワークスペースをパイプラインリソースの代わりに使用できるようになりました。パイプラインリソースはデバッグが容易ではなく、スコープの制限があり、タスクの再利用を可能にしないため、OpenShift Pipelines ではワークスペースを使用することが推奨されます。ワークスペースの詳細は、OpenShift Pipelines のセクションを参照してください。

- ボリューム要求テンプレートのワークスペースのサポートが追加されました。

    - パイプライン実行およびタスク実行のボリューム要求テンプレートがワークスペース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 はパイプラインのすべてのタスク実行の PVC として表示されるテンプレートを使用して永続ボリューム要求 (PVC) を作成します。したがって、複数のタスクにまたがるワークスペース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。

    - ボリューム要求テンプレートがボリュームソースとして使用される場合の PVC 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、パイプラインのすべてのタスク実行のステータスと、パイプライン実行の進捗をモニターするためにパイプライン実行をインスタンス化する際に使用するパイプライン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** リソースのインスタンス化に使用される実際のタスク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各種の状態に適用するサポート。
- クラスタータスクを参照して作成されるタスク実行は、**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。
- kube config writer は、kubeconfig-creator タスク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** 設定をリソース構造に追加できるようになりました。
- **feature-flags** および **config-defaults** 設定マップの名前はカスタマイズ可能になりました。
- タスク実行で使用される Pod テンプレートのホストネットワークのサポートが追加されました。
- Affinity Assistant が、ワークスペースボリュームを共有するタスク実行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。
- Pod テンプレートは、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。
- コントローラーがタスク実行の更新に失敗した場合にタスク実行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。
- Pod テンプレートはタスク実行仕様を使用してランタイム時にタスク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、追加のタスク実行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。
  - パイプライン実行コントローラーは、パイプライン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、タスク実行のクラウドイベント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 などのタスク実行イベントを送信するように設定できます。

- パイプライン実行およびタスク実行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。
- パイプライン実行パラメーターの検証が、パイプラインで必要なすべてのパラメーターがパイプライン実行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、パイプライン実行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。
- パイプライン 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、パイプラインのタスクの失敗後、パイプラインが終了する前に常に実行されるパイプライン内でタスクを指定できるようになりました。
- **git-clone** クラスタータスクが利用できるようになりました。

4.1.12.1.2. 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 つのタスクのみがパイプライン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タスク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、パイプライン実行またはタスク実行のボリューム要求テンプレートを指定できるようになりました。
- **tkn pipelinerun logs** コマンドに、**finally** セクションにリスト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トリガー 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、パイプラインで使用されるタスク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。
- 最近のパイプライン実行またはタスク実行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、パイプラインのタスクに適用される各種の状態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。

4.1.12.1.3. 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。
- イベントリスナーは、**kubectl get** コマンドでリスト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。
- トリガーテンプレートパラメーターは、**$(params.<paramName>)** ではなく **$(tt.params.<paramName>)** 構文を使用するようになり、トリガーテンプレートとリソーステンプレート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イベントリスナー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。
- イベントリスナー Deployment の Readiness Probe を **URL/live** に追加できるようになりました。
- イベントリスナートリガーでの **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。

4.1.12.2. 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、イベントリスナー内の **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)** を使用したトリガーテンプレートの変数の補間が非推奨となり、これにより、パイプライン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** ラベルはクラスタータスク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** クラスタータスクが非推奨になりました。

4.1.12.3. 既知の問題

- 各種の状態はワークスペースには対応しません。
- **--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>)** 構文の後方互換性のサポートにより、トリガーテンプレート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。 トリガー webhook がトリガー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 メトリックは、**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。
- パイプライン実行およびタスク実行は、存在しない PVC 名がワークスペース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。

4.1.12.4. 修正された問題

- 以前のバージョンでは、タスクおよびクラスタータスクの名前が同じ場合、**tkn task delete <name> --trs** コマンドは、タスクとクラスタータスクの両方を削除しました。今回の修正により、コマンドはタスク **<name>** で作成されるタスク実行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 つのパイプライン実行を除きすべてのパイプライン実行を削除しました。今回の修正により、コマンドは最新の 2 つのパイプライン実行を除き、パイプライン **<name>** で作成されるパイプライン実行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リソーステンプレート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** クラスタータスクは、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。

4.1.13. Red Hat OpenShift Pipelines テクノロジープレビュー 1.0 のリリースノート

4.1.13.1. 新機能

Red Hat OpenShift Pipelines 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。Red Hat OpenShift Pipelines 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 をベースとするクラスタータスク

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

4.1.13.1.1. Pipelines

- v1beta1 API バージョンのサポート。

- 改善された制限範囲のサポート。以前のバージョンでは、制限範囲はタスク実行およびパイプライン実行に対してのみ指定されていました。制限範囲を明示的に指定する必要がなくなりました。namespace 間で最小の制限範囲が使用されます。
- タスク結果およびタスクパラメーターを使用してタスク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** 環境変数および各ステップの作業ディレクトリーを上書きしないように設定できるようになりました。
- タスクステップと同様に、**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。
- タスク実行 **podTemplate** リソース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 コントローラーは、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。
- パイプライン、タスク、クラスタータスク、リソース、および状態 (condition) の仕様に新規の説明フィールドが追加されました。
- Git パイプラインリソースへのプロキシーパラメーターの追加。

4.1.13.1.2. 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 **TriggerSBinding**。
- **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**。
- 以下のコマンドは、**--all-namespaces** フラグオプションを使用してすべての namespace からの出力をリスト表示できるようになりました。これらは、**tkn task list**、**tkn pipeline list**、**tkn taskrun list**、**tkn pipelinerun list** です。
  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パイプラインを起動できるようになりました。
- ワークスペースのサポートが **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。

4.1.13.1.3. トリガー

- Trigger は **v1alpha1** および **v1beta1** の両方のパイプライン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。
- イベントリスナーのトリガーレベルでの認証および認可のサポート。

4.1.13.2. 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 **Steps** 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** 変数が **/tekton/home** および **/workspace** 変数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、**disable-home-env-overwrite** および **disable-working-directory-overwrite** フラグを使用して、**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。

4.1.13.3. 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** タスクを複数回送信すると、エラーが発生します。**v1alpha1** タスクの再送信時に、**oc apply** ではなく **oc replace** コマンドを使用します。
- **buildah** クラスタータスクは、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** クラスタータスクの **--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** クラスタータスク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** クラスタータスクを編集します。

```
$ 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** クラスタータスク 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。

4.1.13.4. 修正された問題

- 以前のリリースでは、**DeploymentConfig** タスクは、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、パイプライン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パイプライン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 のパイプライン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パイプラインリソース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、パイプラインが **tekton-pipelines** 以外の namespace にインストールされている場合、**tkn version** コマンドはパイプラインのバージョンを **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、正しいパイプライン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クラスタートリガーバインディングをリスト表示できるようになりました。
- イベントリスナーの **CompareSecret** 機能が、CEL インターセプターについて修正されました。
- タスクおよびクラスタータスクの **list**、**describe**、および **start** サブコマンドは、タスクおよびクラスタータスク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、設定マップを使用して、すべてのタスク実行およびパイプライン実行のタイムアウトが **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 コンソールのパイプライン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。

4.2. OPENSHIFT PIPELINES について

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) が多数導入されています。

4.2.1. 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パイプライン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 Web コンソール **Developer** パースペクティブを使用して、Tekton リソースの作成、パイプライン実行のログの表示、OpenShift Container Platform namespace でのパイプラインの管理を実行できます。

4.2.2. OpenShift Pipelines の概念

本書では、パイプラインの各種概念を詳述します。

4.2.2.1. タスク

Task は Pipeline のビルディングブロックであり、順次実行されるステップで設定されます。これは基本的に入出力の機能です。Task は個別に実行することも、パイプラインの一部として実行することもできます。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。

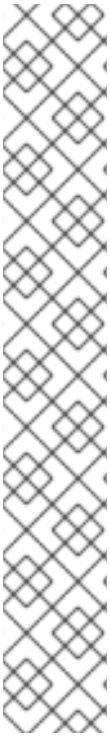
Step は、イメージのビルドなど、Task によって順次実行され、特定の目的を達成するための一連のコマンドです。各タスクは Pod として実行され、各ステップは同じ Pod 内のコンテナーとして実行されます。Step は同じ Pod 内で実行されるため、ファイル、設定マップ、およびシークレットをキャッシュするために同じボリュームにアクセスできます。

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
  workspaces:
  - name: source
  params:
    - name: manifest_dir
      description: The directory in source that contains yaml manifests
      type: string
      default: "k8s"
  steps:
    - name: apply
      image: image-registry.openshift-image-registry.svc:5000/openshift/cli:latest
      workingDir: /workspace/source
      command: ["/bin/bash", "-c"]
      args:
        - |-
          echo Applying manifests in $(params.manifest_dir) directory
          oc apply -f $(params.manifest_dir)
          echo -----------------------------------
```

1. Task API バージョン **v1beta1**。
2. Kubernetes オブジェクトのタイプ **Task**。
3. この Task の一意の名前。
4. Task のパラメーターおよび Step と、Task によって使用される Workspace のリスト。

この Task は Pod を起動し、指定されたコマンドを実行するために指定されたイメージを使用して Pod 内のコンテナーを実行されます。



注記

Pipelines 1.6 以降、この手順の YAML ファイルから、以下のデフォルト設定が削除されます。

- **HOME** 環境変数が **/tekton/home** ディレクトリーにデフォルト設定されない
- **workingDir** フィールドがデフォルトで **/workspace** ディレクトリーにない

代わりに、この手順のコンテナーは **HOME** 環境変数と **workingDir** フィールドを定義します。ただし、この手順の YAML ファイルにカスタム値を指定すると、デフォルト値を上書きできます。

一時的な措置として、古い Pipelines バージョンとの後方互換性を維持するために、**TektonConfig** カスタムリソース定義の以下のフィールドを **false** に設定できます。

```
spec:
  pipeline:
    disable-working-directory-overwrite: false
    disable-home-env-overwrite: false
```

4.2.2.2. when 式

when 式で、パイプライン内のタスクの実行の条件を設定して、タスク実行を保護します。これには、特定の条件が満たされる場合にのみタスクを実行できるようにします。when 式は、パイプライン YAML ファイルの **finally** フィールドを使用して指定される最終タスクセットでもサポートされます。

when 式の主要なコンポーネントは、以下のとおりです。

- **input**: パラメーター、タスクの結果、実行ステータスなどの静的入力または変数を指定します。有効な入力を入力する必要があります。有効な入力を入力しない場合は、デフォルトで空の文字列に設定されます。
- **operator**: **values** セットへの入力の関係を指定します。operator の値として **in** または **notin** を入力します。
- **values**: 文字列値の配列を指定します。ワークスペースに、パラメーター、結果、バインドされたステータスなどの静的値や変数の空でない配列を入力します。

宣言された when 式が、タスクの実行前に評価されます。when 式の値が **True** の場合は、タスクが実行します。when 式の値が **False** の場合、タスクはスキップします。

さまざまなユースケースで when 式を使用できます。たとえば、次のいずれかです。

- 以前のタスクの結果は期待どおりに実行される。
- Git リポジトリーのファイルが以前のコミットで変更になる。
- イメージがレジストリーに存在する。
- 任意のワークスペースが利用可能である。

以下の例は、パイプライン実行の when 式を示しています。パイプライン実行は、次の基準が満たされた場合にのみ **create-file** タスクを実行します。**path** パラメーターが **README.md** です。また、**check-file** タスクから生じる **exists** が **yes** の場合に限り、**echo-file-exists** タスクが実行します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun (1)
metadata:
  generateName: guarded-pr-
spec:
  serviceAccountName: 'pipeline'
  pipelineSpec:
    params:
      - name: path
        type: string
        description: The path of the file to be created
    workspaces:
      - name: source
        description: |
          This workspace is shared among all the pipeline tasks to read/write common resources
    tasks:
      - name: create-file (2)
        when:
          - input: "$(params.path)"
            operator: in
            values: ["README.md"]
        workspaces:
          - name: source
            workspace: source
        taskSpec:
          workspaces:
            - name: source
              description: The workspace to create the readme file in
          steps:
            - name: write-new-stuff
              image: ubuntu
              script: 'touch $(workspaces.source.path)/README.md'
      - name: check-file
        params:
          - name: path
            value: "$(params.path)"
        workspaces:
          - name: source
            workspace: source
        runAfter:
          - create-file
        taskSpec:
          params:
            - name: path
          workspaces:
            - name: source
              description: The workspace to check for the file
          results:
            - name: exists
              description: indicates whether the file exists or is missing
          steps:
            - name: check-file
              image: alpine
              script: |
                if test -f $(workspaces.source.path)/$(params.path); then
                  printf yes | tee /tekton/results/exists
```

```
            else
              printf no | tee /tekton/results/exists
            fi
    - name: echo-file-exists
      when: 3
        - input: "$(tasks.check-file.results.exists)"
          operator: in
          values: ["yes"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: 'echo file exists'
...
    - name: task-should-be-skipped-1
      when: 4
        - input: "$(params.path)"
          operator: notin
          values: ["README.md"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: exit 1
...
  finally:
    - name: finally-task-should-be-executed
      when: 5
        - input: "$(tasks.echo-file-exists.status)"
          operator: in
          values: ["Succeeded"]
        - input: "$(tasks.status)"
          operator: in
          values: ["Succeeded"]
        - input: "$(tasks.check-file.results.exists)"
          operator: in
          values: ["yes"]
        - input: "$(params.path)"
          operator: in
          values: ["README.md"]
      taskSpec:
        steps:
          - name: echo
            image: ubuntu
            script: 'echo finally done'
  params:
    - name: path
      value: README.md
  workspaces:
    - name: source
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
```

```
        storage: 16Mi
```

1. 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。
2. **create-file** タスクが Pipeline で使用されます。
3. **check-file** タスクから生じた **exists** が **yes** になった場合に限り、**echo-file-exists** タスクを実行するのに指定する **when** 式。
4. **path** パラメーターが **README.md** の場合に限り、**task-should-be-skipped-1** タスクをスキップすることを指定する **when** 式。
5. **echo-file-exists** タスクの実行ステータス、およびタスクステータスが **Succeeded** で、**check-file** タスクから生じる **exists** が **yes** になり、**path** パラメーターが **README.md** となる場合に限り、**finally-task-should-be-executed** タスクを実行するのに指定する **when** 式。

OpenShift Container Platform Web コンソールの **Pipeline Run details**ページには、以下のようにタスクと When 式が表示されます。

- すべての基準が満たされています。タスクと、ひし形で表される when 式の記号は緑色です。
- いずれかの基準が満たされていません。タスクはスキップされます。スキップされたタスクと when 式記号は灰色になります。
- 満たされていない基準はありません。タスクはスキップされます。スキップされたタスクと when 式記号は灰色になります。
- タスクの実行が失敗する: 失敗したタスクと when 式の記号が赤で表示されます。

4.2.2.3. 最後のタスク

finally のタスクは、パイプライン YAML ファイルの **finally** フィールドを使用して指定される最終タスクのセットです。**finally** タスクは、パイプライン実行が正常に実行されるかどうかに関係なく、パイプライン内でタスクを常に実行します。**finally** のタスクは、対応するパイプラインが終了する前に、すべてのパイプラインの実行後に並行して実行されます。

同じパイプライン内のタスクの結果を使用するように、**finally** タスクを設定できます。このアプローチでは、この最終タスクが実行される順序は変更されません。これは、最終以外のタスクすべての実行後に他の最終タスクと並行して実行されます。

以下の例は、**clone-cleanup-workspace** パイプラインのコードスニペットを示しています。このコードは、リポジトリーを共有ワークスペースにクローンし、ワークスペースをクリーンアップします。パイプラインタスクの実行後に、パイプライン YAML ファイルの **finally** セクションで指定される **cleanup** タスクがワークスペースをクリーンアップします。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: clone-cleanup-workspace 1
spec:
  workspaces:
    - name: git-source 2
  tasks:
    - name: clone-app-repo 3
      taskRef:
```

```
      name: git-clone-from-catalog
    params:
      - name: url
        value: https://github.com/tektoncd/community.git
      - name: subdirectory
        value: application
    workspaces:
      - name: output
        workspace: git-source
finally:
  - name: cleanup (4)
    taskRef: (5)
      name: cleanup-workspace
    workspaces: (6)
      - name: source
        workspace: git-source
  - name: check-git-commit
    params: (7)
      - name: commit
        value: $(tasks.clone-app-repo.results.commit)
    taskSpec: (8)
      params:
        - name: commit
      steps:
        - name: check-commit-initialized
          image: alpine
          script: |
            if [[ ! $(params.commit) ]]; then
              exit 1
            fi
```

1. Pipeline の一意の名前。
2. git リポジトリーのクローンが作成される共有ワークスペース。
3. アプリケーションリポジトリーを共有ワークスペースにクローンするタスク。
4. 共有ワークスペースをクリーンアップするタスク。
5. TaskRun で実行されるタスクへの参照。
6. 入力を受信し、出力を提供するために 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。
7. タスクに必要なパラメーターのリスト。パラメーターに暗黙的なデフォルト値がない場合は、その値を明示的に設定する必要があります。
8. 埋め込まれたタスク定義。

4.2.2.4. 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、パイプラインの各 Task の 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 で設定され、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、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。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。

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。

4.2.2.5. パイプライン

Pipeline は、特定の実行順序で編成される **Task** リソースのコレクションです。これらは、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ます。1 つ以上のタスクを含むパイプラインを使用して、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設定され、Pipeline が特定の目的を達成することを可能にします。各 **Pipeline** リソース定義には、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。パイプライン定義には、アプリケーション要件に応じて **Conditions**、**Workspaces**、 **Parameters**、または **Resources** をオプションで含めることもできます。

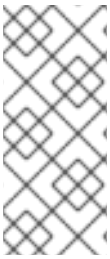
以下の例は、**buildah ClusterTask** を使用して Git リポジトリーからアプリケーションイメージをビルドする **build-and-deploy** パイプラインを示しています。

```
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: "pipelines-1.10"
 - name: IMAGE
   type: string
   description: image to be built from the code
 tasks: 7
 - name: fetch-repository
   taskRef:
     name: git-clone
     kind: ClusterTask
   workspaces:
   - name: output
     workspace: shared-workspace
   params:
   - name: url
     value: $(params.git-url)
   - name: subdirectory
     value: ""
   - name: deleteExisting
     value: "true"
   - name: revision
     value: $(params.git-revision)
 - name: build-image 8
   taskRef:
     name: buildah
     kind: ClusterTask
   params:
   - name: TLSVERIFY
     value: "false"
   - name: IMAGE
     value: $(params.IMAGE)
   workspaces:
   - name: source
     workspace: shared-workspace
   runAfter:
   - fetch-repository
 - name: apply-manifests 9
   taskRef:
     name: apply-manifests
   workspaces:
   - name: source
     workspace: shared-workspace
   runAfter: 10
```

```
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

1. Pipeline API 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 のリストを指定します。
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。



注記

Red Hat OpenShift Pipelines Operator は Buildah クラスタータスクをインストールし、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割り当てて、**パイプライン** サービスアカウントを作成します。Buildah クラスタータスクは、パーミッションが不十分な別のサービスアカウントに関連付けられていると失敗する可能性があります。

4.2.2.6. PipelineRun

PipelineRun は、パイプライン、ワークスペース、認証情報、および CI/CD ワークフローを実行するシナリオ固有のパラメーター値のセットをバインドするリソースタイプです。

pipeline run は、Pipeline の実行中のインスタンスです。これは、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。また、パイプライン実行に、タスクごとのタスク実行も作成します。

パイプラインは、完了するか、タスクが失敗するまでタスクを順次実行します。**status** フィールドは、監視および監査のために、Task run ごとの進捗を追跡し、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: https://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

1. Pipeline Run の API バージョン **v1beta1**。
2. Kubernetes オブジェクトのタイプ。この例では、**PipelineRun** です。
3. この Pipeline Run 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リスト。
6. Pipeline Run で使用する Workspace。

関連情報

- git シークレットを使用したパイプラインの認証

4.2.2.7. Workspaces

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、入力を受信し、出力を提供するために 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。Task または Pipeline は Workspace を宣言し、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。その後、これは TaskRun または PipelineRun の Workspace にマウントされます。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 のリスト。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 のリスト。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 のリスト。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。

Workspace はタスクがデータを共有する際に使用でき、これにより、パイプラインの各タスク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。永続ボリューム要求 (PVC) を作成するか、永続ボリューム要求 (PVC) を作成するボリューム要求テンプレートを指定できます。

以下の **build-deploy-api-pipelinerun** PipelineRun のコードスニペットは、**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の永続ボリューム要求 (PVC) を作成するために永続ボリュームテンプレートを使用します。

```
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
    volumeClaimTemplate: 3
      spec:
```

```
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi
```

1. PipelineRun にボリュームバインディングを提供する Pipeline Workspace のリストを指定します。
2. ボリュームが提供されている Pipeline の Workspace の名前。
3. ワークスペースのストレージボリュームを定義するために永続ボリューム要求 (PVC) を作成するボリューム要求テンプレートを指定します。

4.2.2.8. トリガー

Trigger をパイプラインと併用して、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。Trigger は、Git プルリクエストなどの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 リソースを作成およびデプロイし、パイプラインをインスタンス化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、パイプラインを開始する必要があります。トリガー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パイプライン実行をトリガーして、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リソースで設定されています。

- **TriggerBinding** リソースは、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。
  以下の例は、**TriggerBinding** リソース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/v1beta1 1
kind: TriggerBinding 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

1. **TriggerBinding** リソースの API バージョン。この例では、**v1beta1** です。
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リス

- **TriggerTemplate** リソースは、リソースの作成方法の標準として機能します。これは、**TriggerBinding** リソースからのパラメーター化されたデータが使用される方法を指定します。トリガーテンプレートは、トリガーバインディングから入力を受信し、新規パイプラインリソースの作成および新規パイプライン実行の開始につながる一連のアクションを実行します。
  以下の例は、**TriggerTemplate** リソースのコードスニペットを示しています。これは、作成した **TriggerBinding** リソースから受信される Git リポジトリー情報を使用してパイプライン実行を作成します。

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: TriggerTemplate 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.10
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(tt.params.git-repo-name)-$(uid)
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/$(tt.params.git-repo-name)
      workspaces:
      - name: shared-workspace
        volumeClaimTemplate:
          spec:
            accessModes:
             - ReadWriteOnce
            resources:
              requests:
                storage: 500Mi
```

1. **TriggerTemplate** リソースの API バージョン。この例では、**v1beta1** です。
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. **TriggerTemplate** リソースを識別するための一意の名前。
4. **TriggerBinding** リソースによって提供されるパラメーター。
5. **TriggerBinding** または **EventListener** リソースを使用して受信されるパラメーターを使用してリソースを作成する必要がある方法を指定するテンプレートの一覧。

- **Trigger** リソースは、**TriggerBinding** リソースおよび **TriggerTemplate** リソースと、オプションで **interceptors** イベントプロセッサーを組み合わせます。
インターセプターは、**TriggerBinding** リソースの前に実行される特定プラットフォームのすべてのイベントを処理します。インターセプターを使用して、ペイロードのフィルタリング、イベントの検証、トリガー条件の定義およびテスト、および他の有用な処理を実装できます。インターセプターは、イベント検証にシークレットを使用します。イベントデータがインターセプターを通過したら、ペイロードデータをトリガーバインディングに渡す前にトリガーに移動します。インターセプターを使用して、**EventListener** 仕様で参照される関連付けられたトリガーの動作を変更することもできます。

  以下の例は、**TriggerBinding** および **TriggerTemplate** リソースを接続する **vote-trigger** という名前の **Trigger** リソースのコードスニペットと、**interceptors** イベントプロセッサーを示しています。

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: Trigger 2
metadata:
  name: vote-trigger 3
spec:
  serviceAccountName: pipeline 4
  interceptors:
    - ref:
        name: "github" 5
      params: 6
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
    - ref: vote-app 7
  template: 8
    ref: vote-app
---
apiVersion: v1
kind: Secret 9
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "1234567"
```

1. **Trigger** リソースの API バージョン。この例では、**v1beta1** です。
2. Kubernetes オブジェクトのタイプを指定します。この例では、**Trigger** です。

3. この **Trigger** リソースを識別するための一意の名前。
4. 使用されるサービスアカウント名。
5. 参照されるインターセプター名。この例では、**github** です。
6. 指定する必要のあるパラメーター。
7. **TriggerTemplate** リソースに接続する **TriggerBinding** リソースの名前。
8. **TriggerBinding** リソースに接続するための **TriggerTemplate** リソースの名前。
9. イベントの検証に使用されるシークレット。

- **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。これは各 **TriggerBinding** リソースからイベントパラメーターを抽出し、次にこのデータを処理し、対応する **TriggerTemplate** リソースによって指定される Kubernetes リソースを作成します。**EventListener** リソースは、イベントの **interceptors** を使用してペイロードで軽量イベント処理または基本的なフィルターを実行します。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。現時点で、パイプライントリガーは **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**、**Bitbucket 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。
以下の例は、**vote-trigger** という名前の **Trigger** リソースを参照する **EventListener** リソースを示しています。

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: EventListener 2
metadata:
  name: vote-app 3
spec:
  serviceAccountName: pipeline 4
  triggers:
    - triggerRef: vote-trigger 5
```

1. **EventListener** リソースの API バージョン。この例では、**v1beta1** です。
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. **EventListener** リソースを識別するための一意の名前。
4. 使用されるサービスアカウント名。
5. **EventListener** リソースによって参照される **Trigger** リソースの名前。

4.2.3. 関連情報

- パイプラインのインストールについての詳細は、Installing OpenShift Pipelines 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、Creating applications with CI/CD Pipelines を参照してください。
- re-encrypt TLS 終端の詳細は、再暗号化終端 を参照してください。

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

4.3. OPENSHIFT PIPELINES のインストール

以下では、クラスター管理者を対象に、Red Hat OpenShift Pipelines Operator の OpenShift Container Platform クラスターへのインストールプロセスについて説明します。

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。
- OpenShift Pipelines (**tkn**) CLI がローカルシステムにインストールされている。

4.3.1. Web コンソールでの Red Hat OpenShift Pipelines Operator のインストール

OpenShift Container Platform OperatorHub にリスト表示されている 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、パイプラインの設定に必要なカスタムリソース (CR) が Operator と共に自動的にインストールされます。

デフォルトの Operator カスタムリソース定義 (CRD) の **config.operator.tekton.dev** が **tektonconfigs.operator.tekton.dev** に置き換えられました。さらに Operator は、個別に管理される OpenShift Pipelines コンポーネントに追加の CRD (**tektonpipelines.operator.tekton.dev**、**tektontriggers.operator.tekton.dev** および **tektonaddons.operator.tekton.dev**) を提供します。

OpenShift Pipelines がクラスターにすでにインストールされている場合、既存のインストールはシームレスにアップグレードされます。Operator は必要に応じて、クラスターの **config.operator.tekton.dev** のインスタンスを **tektonconfigs.operator.tekton.dev** のインスタンスと、その他の CRD の追加オブジェクトに置き換えます。

警告

既存のインストールを手動で変更した場合 (**resource name - cluster** フィールドに変更を加えて **config.operator.tekton.dev** CRD インスタンスのターゲット namespace を変更する場合など)、アップグレードパスはスムーズではありません。このような場合は、インストールをアンインストールし、Red Hat OpenShift Pipelines Operator を再インストールするワークフローが推奨されます。

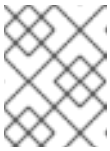
Red Hat OpenShift Pipelines Operator は、**TektonConfig** CR の一部としてプロファイルを指定して、インストールするコンポーネントを選択するオプションを提供するようになりました。**TektonConfig** CR は Operator のインストール時に自動的にインストールされます。サポートされるプロファイルは以下のとおりです。

- Lite: これは Tekton パイプラインのみをインストールします。
- Basic: これは Tekton パイプラインと Tekton トリガーをインストールします。

- All: これは **TektonConfig** CR のインストール時に使用されるデフォルトプロファイルです。このプロファイルは、Tekton Pipelines、Tekton Triggers、Tekton Addons (**ClusterTasks**、**ClusterTriggerBindings**、**ConsoleCLIDownload**、**ConsoleQuickStart** および **ConsoleYAMLSample** リソースを含む) のすべてをインストールします。

手順

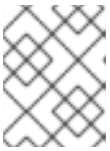
1. Web コンソールの **Administrator** パースペクティブで、**Operators → OperatorHub** に移動します。
2. **Filter by keyword** ボックスを使用して、カタログで **Red Hat OpenShift Pipelines** Operator を検索します。**Red Hat OpenShift Pipelines** Operator タイルをクリックします。
3. **Red Hat OpenShift Pipelines** Operator ページで Operator についての簡単な説明を参照してください。**Install** 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** で **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 に Operator をインストールします。これにより、Operator はクラスター内のすべての namespace を監視し、これらの namespace に対して利用可能になります。
   b. **Approval Strategy** で **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。
      - **pipelines-<version>** チャネルは、Red Hat OpenShift Pipelines Operator をインストールするためのデフォルトのチャネルです。たとえば、Red Hat OpenShift Pipelines Operator バージョン **1.7** をインストールするためのデフォルトのチャネルは **pipelines-1.7** です。
      - **latest** チャネルにより、Red Hat OpenShift Pipelines Operator の最新の安定バージョンをインストールできます。



注記

preview チャネルと **stable** チャネルは廃止され、将来のリリースで削除される予定です。

5. **Install** をクリックします。Operator が **Installed Operators** ページにリスト表示されます。



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。

警告

他のコンポーネントのインストールが進行中の場合でも、成功ステータスが **Succeeded Up to date** として表示される場合があります。したがって、ターミナルで手動でインストールを確認することが重要です。

7. Red Hat OpenShift Pipelines Operator のすべてのコンポーネントが正常にインストールされたことを確認します。ターミナルでクラスターにログインし、次のコマンドを実行します。

```
$ oc get tektonconfig config
```

出力例

```
NAME     VERSION   READY   REASON
config   1.9.2     True
```

READY 条件が **True** の場合、Operator とそのコンポーネントは正常にインストールされています。

さらに、次のコマンドを実行して、コンポーネントのバージョンを確認します。

```
$ oc get tektonpipeline,tektontrigger,tektonaddon,pac
```

出力例

```
NAME                                    VERSION   READY   REASON
tektonpipeline.operator.tekton.dev/pipeline   v0.41.1   True
NAME                                   VERSION   READY   REASON
tektontrigger.operator.tekton.dev/trigger   v0.22.2   True
NAME                                VERSION   READY   REASON
tektonaddon.operator.tekton.dev/addon   1.9.2     True
NAME                                                VERSION   READY   REASON
openshiftpipelinesascode.operator.tekton.dev/pipelines-as-code   v0.15.5   True
```

4.3.2. 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 のチャネル名。デフォルトチャネルは **pipelines-<version>** です。たとえば、Red Hat OpenShift Pipelines Operator バージョン **1.7** のデフォルトチャネルは **pipelines-1.7** です。**latest** チャネルにより、Red Hat OpenShift Pipelines Operator の最新の stable バージョンがインストール可能になります。
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
```

これで 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ました。

4.3.3. 制限された環境での Red Hat OpenShift Pipelines Operator

Red Hat OpenShift Pipelines Operator は、ネットワークが制限された環境でのパイプラインのインストールに対するサポートを有効にします。

Operator は、**cluster** プロキシーオブジェクトに基づいて tekton-controllers によって作成される Pod のコンテナーにプロキシー環境変数を設定するプロキシー Webhook をインストールします。また、プロキシー環境変数を **TektonPipelines**、**TektonTriggers**、**Controllers**、**Webhooks**、および **Operator Proxy Webhook** リソースに設定します。

デフォルトで、プロキシー Webhook は **openshift-pipelines** namespace に対して無効にされます。他の namespace に対してこれを無効にするには、**operator.tekton.dev/disable-proxy: true** ラベルを **namespace** オブジェクトに追加します。

4.3.4. 関連情報

- Operator の OpenShift Container Platform へのインストール方法については、adding Operators to a cluster セクションを参照してください。
- Red Hat OpenShift Pipelines Operator を使用して Tekton Chains をインストールするには、Using Tekton Chains for Red Hat OpenShift Pipelines supply chain security を参照してください。
- クラスター内の Tekton Hub をインストールしてデプロイするには、Red Hat OpenShift Pipeline での Tekton Hub の使用 を参照してください。
- 制限された環境でパイプラインを使用する方法についての詳細は、以下を参照してください。

- 制限された環境でパイプラインを実行するためのイメージのミラーリング
- 制限されたクラスターの Samples Operator の設定
- ミラーリングされたレジストリーでのクラスターの作成

4.4. OPENSHIFT PIPELINES のアンインストール

クラスター管理者は、以下のステップを実行することにより、Red Hat OpenShift Pipelines Operator をアンインストールできます。

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. Operator に依存する Tekton Hub などのオプションコンポーネントの CR を削除します。

   注意

   オプションコンポーネントの CR を削除せずに Operator をアンインストールした場合、後で削除できません。

3. 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。

4.4.1. 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

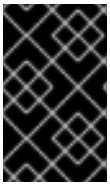
1. Web コンソールの **Administrator** パースペクティブで、**Administration → Custom Resource Definition** に移動します。
2. **Filter by name** ボックスに **config.operator.tekton.dev** を入力し、 Red Hat OpenShift Pipelines Operator CR を検索します。
3. **CRD Config** をクリックし、**Custom Resource Definition Details**ページを表示します。
4. **Actions** 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



重要

Operator をアンインストールする前に、この手順を繰り返して Tekton Hub などのオプションコンポーネントの CR を見つけ、削除します。オプションコンポーネントの CR を削除せずに Operator をアンインストールした場合、後で削除できません。

4.4.2. Red Hat OpenShift Pipelines Operator のアンインストール

Web コンソールの **Administrator** パースペクティブを使用して、Red Hat OpenShift Pipelines Operator をアンインストールできます。

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. **Red Hat OpenShift Pipelines** Operator タイルをクリックします。Operator タイルは、Operator がインストールされていることを示します。
3. **Red Hat OpenShift Pipelines** Operator の説明ページで、**Uninstall** をクリックします。

関連情報

- Operator の OpenShift Container Platform でのアンインストール方法は、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。

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

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

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

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

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

- **pipelines-vote-ui** Git リポジトリーにソースコードがあるフロントエンドインターフェイス (**pipelines-vote-ui**)。

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

4.5.1. 前提条件

- OpenShift Container Platform クラスターにアクセスできる。
- OpenShift OperatorHub に一覧表示されている Red Hat OpenShift Pipelines Operator を使用して OpenShift Pipelines をインストールしている。インストールの完了後にクラスター全体に適用できる。
- OpenShift Pipelines CLI をインストールしている。
- GitHub ID を使用してフロントエンドの **pipelines-vote-ui** およびバックエンドの **pipelines-vote-api** Git リポジトリーをフォークしており、これらのリポジトリーに管理者権限でアクセスできる。
- オプション: **pipelines-tutorial** Git リポジトリーのクローンを作成している。

4.5.2. プロジェクトの作成およびパイプラインのサービスアカウントの確認

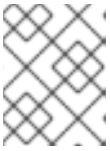
手順

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

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

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

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



注記

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

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

```
$ oc get serviceaccount pipeline
```

4.5.3. パイプラインタスクの作成

手順

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/02_update_deployment_task.yaml
```

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

```
$ tkn task list
```

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

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

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

注記

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

```
$ tkn clustertasks list
```

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

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

関連情報

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

4.5.4. パイプラインのアセンブル

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

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

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

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

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

手順

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

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
  - name: shared-workspace
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "pipelines-1.10"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
```

```
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

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

2. パイプラインを作成します。

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/01_pipeline/04_pipeline.yaml
```

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

```
$ tkn pipeline list
```

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

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

-

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

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

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

手順

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

   a. 必要なイメージタグがインポートされていないことを確認します。

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

   出力例

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

   3.8-ubi8 (latest)
     tagged from registry.redhat.io/ubi8/python-38:latest
       prefer registry pullthrough when referencing this tag

     Build and run Python 3.8 applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-python-container/blob/master/3.8/README.md.
     Tags: builder, python
     Supports: python:3.8, python
     Example Repo: https://github.com/sclorg/django-ex.git

   [...]
   ```

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

   ```
   $ oc image mirror registry.redhat.io/ubi8/python-38:latest <mirror-registry>:<port>/ubi8/python-38
   ```

   c. イメージをインポートします。

   ```
   $ oc tag <mirror-registry>:<port>/ubi8/python-38 python:latest --scheduled -n openshift
   ```

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

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

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

出力例

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

latest
  updates automatically from registry <mirror-registry>:<port>/ubi8/python-38

  * <mirror-registry>:<port>/ubi8/python-
38@sha256:3ee3c2e70251e75bfeac25c0c33356add9cc4abcbc9c51d858f39e4dc29c5f58


[...]
```

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

   a. 必要なイメージタグがインポートされていないことを確認します。

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

   出力例

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

   1.14.7-ubi8 (latest)
     tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
       prefer registry pullthrough when referencing this tag

     Build and run Go applications on UBI 8. For more information about using this builder
   image, including OpenShift considerations, see https://github.com/sclorg/golang-
   container/blob/master/README.md.
     Tags: builder, golang, go
     Supports: golang
     Example Repo: https://github.com/sclorg/golang-ex.git

   [...]
   ```

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

   ```
   $ oc image mirror registry.redhat.io/ubi8/go-toolset:1.14.7 <mirror-registry>:
   <port>/ubi8/go-toolset
   ```

   c. イメージをインポートします。

   ```
   $ oc tag <mirror-registry>:<port>/ubi8/go-toolset golang:latest --scheduled -n openshift
   ```

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

d. 指定されたタグを持つイメージがインポートされていることを確認します。

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

出力例

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

latest
  updates automatically from registry <mirror-registry>:<port>/ubi8/go-toolset

  * <mirror-registry>:<port>/ubi8/go-
toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421
b37

[...]
```

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

a. 必要なイメージタグがインポートされていないことを確認します。

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

出力例

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

latest
  updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


  * quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

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

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
<mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

c. イメージをインポートします。

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

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

d. 指定されたタグを持つイメージがインポートされていることを確認します。

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

出力例

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

latest
  updates automatically from registry <mirror-registry>:<port>/openshift-release-dev/ocp-
v4.0-art-dev

  * <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551


[...]
```

関連情報

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

4.5.6. パイプラインの実行

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

手順

1. バックエンドアプリケーションのパイプラインを起動します。

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-api \
    -p git-url=https://github.com/openshift/pipelines-vote-api.git \
    -p IMAGE='image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/pipelines-vote-api' \
    --use-param-defaults
```

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

2. パイプライン実行の進捗を追跡するには、以下のコマンドを入力します。

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

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

3. フロントエンドアプリケーションのパイプラインを起動します。

```
$ tkn pipeline start build-and-deploy \
    -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.10/01_pipeline/03_persistent_volume_claim.yaml \
    -p deployment-name=pipelines-vote-ui \
    -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
    -p IMAGE='image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/pipelines-vote-ui' \
    --use-param-defaults
```

4. パイプライン実行の進捗を追跡するには、以下のコマンドを入力します。

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

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

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

```
$ tkn pipelinerun list
```

出力には、パイプライン実行がリスト表示されます。

```
NAME                      STARTED      DURATION     STATUS
build-and-deploy-run-xy7rw   1 hour ago   2 minutes    Succeeded
build-and-deploy-run-z2rz8   1 hour ago   19 minutes   Succeeded
```

6. アプリケーションルートを取得します。

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

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

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

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

関連情報

- git シークレットを使用したパイプラインの認証

4.5.7. トリガーのパイプラインへの追加

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

手順

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

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

2. **TriggerBinding** リソースを作成します。

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/03_triggers/01_binding.yaml
```

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

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
  params:
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.10
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      generateName: build-deploy-$(tt.params.git-repo-name)-
```

```
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/$(tt.params.git-repo-name)
      workspaces:
      - name: shared-workspace
        volumeClaimTemplate:
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 500Mi
```

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

4. **TriggerTemplate** リソースを作成します。

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-
1.10/03_triggers/02_template.yaml
```

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

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

6. **Trigger** リソースを作成します。

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/03_triggers/03_trigger.yaml
```

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

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

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

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

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

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

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

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

  b. **EventListener** リソースを作成します。

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

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

  ```
  $ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.10/03_triggers/04_event_listener.yaml
  ```

  c. re-encrypt TLS 終端でルートを作成します。

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

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

セキュアなルートの re-encrypt TLS 終端 YAML の例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend 2
  tls:
    termination: reencrypt 3
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- 4
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

1 2 オブジェクトの名前で、63 文字に制限されます。

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

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

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

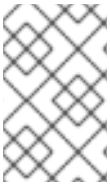
- 非セキュアな HTTP 接続を使用して **EventListener** リソースを作成するには、以下を実行します。

  a. **EventListener** リソースを作成します。

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

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

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



注記

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

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

手順

1. イベントリスナーのクラスター全体のフェッチ権限を設定します。

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

   ServiceAccount.yaml の例

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

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

   ClusterRole.yaml の例

   ```
   kind: ClusterRole
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
     name: el-sel-clusterrole
   rules:
   - apiGroups: ["triggers.tekton.dev"]
     resources: ["eventlisteners", "clustertriggerbindings", "clusterinterceptors",
   "triggerbindings", "triggertemplates", "triggers"]
     verbs: ["get", "list", "watch"]
   - apiGroups: [""]
     resources: ["configmaps", "secrets"]
     verbs: ["get", "list", "watch"]
   - apiGroups: [""]
     resources: ["serviceaccounts"]
     verbs: ["impersonate"]
   ...
   ```

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

   ClusterRoleBinding.yaml の例

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   ```

```
  metadata:
    name: el-mul-clusterrolebinding
  subjects:
  - kind: ServiceAccount
    name: el-sa
    namespace: default
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: el-sel-clusterrole
  ...
```

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

EventListener.yaml の例

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

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

ServiceAccount.yaml の例

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

RoleBinding.yaml の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggercr-rolebinding
  namespace: foo
subjects:
- kind: ServiceAccount
  name: foo-trigger-sa
  namespace: foo
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
...
```

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

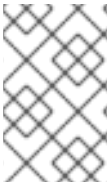
Trigger.yaml の例

```
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  serviceAccountName: foo-trigger-sa
  interceptors:
    - ref:
        name: "github"
      params:
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
...
```

4.5.9. Webhook の作成

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

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



注記

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

手順

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

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

- HTTP (非セキュアな) 接続の場合:

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

出力で取得した URL をメモします。

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

4.5.10. パイプライン実行のトリガー

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

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

手順

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

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.10
```

2. 空のコミットをプッシュします。

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.10
```

3. パイプライン実行がトリガーされたかどうかを確認します。

```
$ tkn pipelinerun list
```

新規のパイプライン実行が開始されたことに注意してください。

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

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

前提条件

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

手順

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

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/managed-by: EventListener
    app.kubernetes.io/part-of: Triggers
    eventlistener: github-listener
  annotations:
    networkoperator.openshift.io/ignore-errors: ""
  name: el-monitor
  namespace: test
spec:
  endpoints:
    - interval: 10s
      port: http-metrics
  jobLabel: name
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: EventListener
      app.kubernetes.io/part-of: Triggers
      eventlistener: github-listener
...
```

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

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

3. OpenShift Container Platform Web コンソールで、**Administrator → Observe → Metrics** の順に移動します。

4. メトリックを表示するには、名前で検索します。たとえば、**github-listener** イベントリスナーの **eventlistener_http_resources** メトリックの詳細を表示するには、**eventlistener_http_resources** のキーワードを使用して検索します。

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

4.5.12. 関連情報

- Pipelines as Code とアプリケーションソースコードを同一レポジトリーに格納するには、Pipelines as Code の使用 を参照してください。
- **Developer** パースペクティブのパイプラインの詳細は、Web コンソールでのパイプラインの使用 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。
- 再利用可能なタスクの追加の例については、OpenShift Catalog リポジトリーを参照してください。さらに、Tekton プロジェクトで Tekton Catalog を参照することもできます。
- 再利用可能なタスクとパイプライン用に Tekton Hub のカスタムインスタンスをインストールしてデプロイするには、Red Hat OpenShift Pipeline での Tekton Hub の使用 を参照してください。
- re-encrypt TLS 終端の詳細は、再暗号化終端 を参照してください。
- セキュリティー保護されたルートの詳細は、Secured routes セクションを参照してください。

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

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

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

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

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

表4.5 バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの違い

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

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

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

表4.6 バージョン付けされていないクラスタータスクとバージョン付けされたクラスタータスクの長所と短所

クラスタータスク	メリット	デメリット
バージョン付けされていないクラスタータスク (NVCT)	• 最新の更新およびバグ修正でパイプラインをデプロイする場合は、NVCT を使用します。 • Operator をアップグレードすると、バージョン付けされていないクラスタータスクがアップグレードされます。これは、複数のバージョン付けされたクラスタータスクよりも少ないリソースを消費します。	NVCT を使用するパイプラインをデプロイする場合、自動的にアップグレードされたクラスタータスクが後方互換性を持たない場合、Operator のアップグレード後にそれらが破損する可能性があります。

クラスタータスク	メリット	デメリット
バージョン付けされたクラスタータスク (VCT)	● 実稼働で安定したパイプラインが重要視される場合は、VCT を使用します。 ● 新しいバージョンのクラスタータスクがインストールされた後でも、以前のバージョンはクラスターで保持されます。以前のクラスタータスクを引き続き使用できます。	● 以前のバージョンのクラスタータスクを引き続き使用する場合は、最新の機能と重要なセキュリティー更新が欠落している可能性があります。 ● 動作していない以前のバージョンのクラスタータスクがクラスターリソースを消費します。 ● * アップグレード後に、Operator は以前の VCT を管理できません。**oc delete clustertask** コマンドを使用して、以前の VCT を手動で削除できますが、復元することはできません。

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

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

手順

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

   TektonConfig CR の例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
  - name: createRbacResource
    value: "false"
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
    - name: clusterTasks
      value: "false"
...
```

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

注記

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

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

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

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

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

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

   注意

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

4.7. OPENSHIFT PIPELINE での TEKTON HUB の使用

重要

Tekton Hub はテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

Tekton Hub は、CI/CD ワークフローの再利用可能なタスクとパイプラインを検出、検索、および共有するのに役立ちます。Tekton Hub のパブリックインスタンスは、hub.tekton.dev で利用できます。クラスター管理者は、エンタープライズで使用するために Tekton Hub のカスタムインスタンスをインストールしてデプロイすることもできます。

4.7.1. OpenShift Container Platform クラスターへの Tekton Hub のインストールとデプロイ

Tekton Hub はオプションのコンポーネントです。クラスター管理者は、**TektonConfig** カスタムリソース (CR) を使用してこれをインストールできません。Tekton Hub をインストールおよび管理するには、**TektonHub** CR を使用します。

注記

Github Enterprise または Gitlab Enterprise を使用している場合は、エンタープライズサーバーと同じネットワークに Tekton Hub をインストールしてデプロイします。たとえば、エンタープライズサーバーが VPN の背後で実行されている場合は、同じく VPN の背後にあるクラスターに Tekton Hub をデプロイします。

前提条件

- Red Hat OpenShift Pipelines Operator が、クラスターのデフォルトの **openshift-pipelines** namespace にインストールされている。

手順

1. Tekton Hub リポジトリーのフォークを作成します。
2. フォークされたリポジトリーのクローンを作成します。
3. **config.yaml** ファイルを更新して、次のスコープを持つ少なくとも 1 人のユーザーを含めます。
   - **agent:create** スコープを持つユーザーで、カタログに変更があった場合に、一定間隔後に Tekton Hub データベースを更新する cron ジョブを設定できます。
   - Tekton Hub のデータベース内のカタログとすべてのリソースを更新できる **catalog:refresh** スコープを持つユーザー。
   - 追加のスコープを取得できる **config:refresh** スコープを持つユーザー。

     ```
     ...
     scopes:
     - name: agent:create
       users: <username_registered_with_the_Git_repository_hosting_service_provider>
     - name: catalog:refresh
       users: <username_registered_with_the_Git_repository_hosting_service_provider>
     - name: config:refresh
       users: <username_registered_with_the_Git_repository_hosting_service_provider>
     ...
     ```

     サポートされているサービスプロバイダーは、GitHub、GitLab、および BitBucket です。
4. Git リポジトリーホスティングプロバイダーを使用して OAuth アプリケーションを作成し、クライアント ID とクライアントシークレットをメモします。
   - GitHub OAuth アプリケーションの場合、**Homepage URL** と **Authorization callback URL** を **<auth-route>** として設定します。
   - GitLab OAuth アプリケーションの場合、**REDIRECT_URI** を **<auth-route>/auth/gitlab/callback** として設定します。
   - BitBucket OAuth アプリケーションの場合、**Callback URL** を **<auth-route>** として設定します。

5. Tekton Hub API シークレットの **<tekton_hub_repository>/config/02-api/20-api-secret.yaml** ファイルの次のフィールドを編集します。

   - **GH_CLIENT_ID**: Git リポジトリーホスティングサービスプロバイダーで作成された OAuth アプリケーションのクライアント ID。
   - **GH_CLIENT_SECRET**: Git リポジトリーホスティングサービスプロバイダーで作成された OAuth アプリケーションのクライアントシークレット。
   - **GHE_URL**: GitHub Enterprise を使用して認証している場合は、GitHub Enterprise URL。このフィールドの値としてカタログへの URL を指定しないでください。
   - **GL_CLIENT_ID**: GitLab OAuth アプリケーションからのクライアント ID。
   - **GL_CLIENT_SECRET**: GitLab OAuth アプリケーションからのクライアントシークレット。
   - **GLE_URL**: GitLab Enterprise を使用して認証している場合は、GitLab Enterprise URL。このフィールドの値としてカタログへの URL を指定しないでください。
   - **BB_CLIENT_ID**: BitBucket OAuth アプリケーションからのクライアント ID。
   - **BB_CLIENT_SECRET**: BitBucket OAuth アプリケーションからのクライアントシークレット。
   - **JWT_SIGNING_KEY**: ユーザー用に作成された JSON Web Token (JWT) に署名するために使用される長いランダムな文字列。
   - **ACCESS_JWT_EXPIRES_IN**: アクセストークンの有効期限が切れるまでの制限時間を追加します。たとえば、**1m**、ここで **m** は分を示します。サポートされている時間の単位は、秒 (**s**)、分 (**m**)、時間 (**h**)、日 (**d**)、および週 (**w**) です。
   - **REFRESH_JWT_EXPIRES_IN**: 更新トークンの有効期限が切れるまでの制限時間を追加します。たとえば、**1m**、ここで **m** は分を示します。サポートされている時間の単位は、秒 (**s**)、分 (**m**)、時間 (**h**)、日 (**d**)、および週 (**w**) です。トークンの更新に設定された有効期限が、トークンアクセスに設定された有効期限よりも長いことを確認してください。
   - **AUTH_BASE_URL**: OAuth アプリケーションのルート URL。

   注記

   - サポートされている Git リポジトリーホスティングサービスプロバイダーのいずれかについて、クライアント ID とクライアントシークレットに関連するフィールドを使用します。
   - Git リポジトリーホスティングサービスプロバイダーに登録されたアカウントクレデンシャルにより、**catalog: refresh** スコープを使用するユーザーは、すべてのカタログリソースを認証してデータベースにロードできます。

6. 変更をコミットして、フォークされたリポジトリーにプッシュします。

7. **TektonHub** CR が次の例のようになっていることを確認します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonHub
```

```
metadata:
  name: hub
spec:
  targetNamespace: openshift-pipelines (1)
  api:
    hubConfigUrl: https://raw.githubusercontent.com/tektoncd/hub/main/config.yaml (2)
```

(1) Tekton Hub をインストールする必要がある namespace。デフォルトは **openshift-pipelines** です。

(2) フォークされたリポジトリーの **config.yaml** ファイルの URL に置き換えます。

8. Tekton Hub をインストールします。

```
$ oc apply -f TektonHub.yaml (1)
```

(1) **TektonConfig** CR のファイル名またはパス。

9. インストールのステータスを確認します。

```
$ oc get tektonhub.operator.tekton.dev
NAME   VERSION   READY   REASON   APIURL                 UIURL
hub    v1.7.2    True             https://api.route.url/    https://ui.route.url/
```

4.7.1.1. Tekton Hub でカタログを手動で更新する

OpenShift Container Platform クラスターに Tekton Hub をインストールしてデプロイすると、Postgres データベースもインストールされます。最初、データベースは空です。カタログで使用可能なタスクとパイプラインをデータベースに追加するには、クラスター管理者はカタログを更新する必要があります。

前提条件

- **<tekton_hub_repository>/config/** ディレクトリーにいることを確認してください。

手順

1. Tekton Hub UI で、**Login -→ Sign In With GitHub**をクリックします。

注記

GitHub は、公開されている Tekton Hub UI の例として使用されています。クラスターへのカスタムインストールの場合、クライアント ID とクライアントシークレットを提供したすべての Git リポジトリーホスティングサービスプロバイダーがリスト表示されます。

2. ホームページで、ユーザープロファイルをクリックし、トークンをコピーします。
3. カタログ更新 API を呼び出します。
   - 特定の名前でカタログを更新するには、次のコマンドを実行します。

```
$ curl -X POST -H "Authorization: <jwt-token>" \ 1
  <api-url>/catalog/<catalog_name>/refresh 2
```

1. UI からコピーされた Tekton Hub トークン。
2. API Pod の URL とカタログの名前。

出力サンプル

```
[{"id":1,"catalogName":"tekton","status":"queued"}]
```

- すべてのカタログを更新するには、次のコマンドを実行します。

```
$ curl -X POST -H "Authorization: <jwt-token>" \ 1
  <api-url>/catalog/refresh 2
```

1. UI からコピーされた Tekton Hub トークン
2. API Pod の URL。

4. ブラウザーでページを更新します。

4.7.1.2. オプション: Tekton Hub でカタログを更新するための cron ジョブの設定

クラスター管理者は、オプションで cron ジョブを設定して、一定の間隔の後にデータベースを更新し、カタログの変更が Tekton Hub Web コンソールに表示されるようにすることができます。

注記

リソースがカタログに追加または更新された場合、カタログを更新すると、これらの変更が Tekton Hub UI に表示されます。ただし、リソースがカタログから削除された場合、カタログを更新してもデータベースからリソースは削除されません。Tekton Hub UI は、削除されたリソースを引き続き表示します。

前提条件

- **<project_root>/config/** ディレクトリーにいることを確認します。ここで、**<project_root>** は、複製された Tekton Hub リポジトリーの最上位ディレクトリーです。
- カタログを更新するスコープを持つ JSON Web トークン (JWT) トークンがあることを確認します。

手順

1. 長期間使用するためのエージェントベースの JWT トークンを作成します。

```
$ curl -X PUT --header "Content-Type: application/json" \
    -H "Authorization: <access-token>" \ 1
    --data '{"name":"catalog-refresh-agent","scopes": ["catalog:refresh"]}' \
    <api-route>/system/user/agent
```

1 JWT トークン。

必要なスコープを持つエージェントトークンは、**{"token":"<agent_jwt_token>"}** 形式で返されます。返されたトークンをメモし、カタログ更新 cron ジョブ用に保存します。

2. **05-catalog-refresh-cj/50-catalog-refresh-secret.yaml** ファイルを編集して、**HUB_TOKEN** パラメーターを前の手順で返された **<agent_jwt_token>** に設定します。

```
apiVersion: v1
kind: Secret
metadata:
  name: catalog-refresh
type: Opaque
stringData:
  HUB_TOKEN: <hub_token> 1
```

1 前の手順で返された **<agent_jwt_token>**。

3. 変更した YAML ファイルを適用します。

```
$ oc apply -f 05-catalog-refresh-cj/ -n openshift-pipelines.
```

4. オプション: デフォルトでは、cron ジョブは 30 分ごとに実行するように設定されています。間隔を変更するには、**05-catalog-refresh-cj/51-catalog-refresh-cronjob.yaml** ファイルの **schedule** パラメーターの値を変更します。

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: catalog-refresh
  labels:
    app: tekton-hub-api
spec:
  schedule: "*/30 * * * *"
  ...
```

4.7.1.3. オプション: Tekton Hub に設定に新しいユーザーを追加する

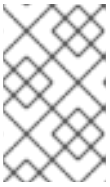
手順

1. 目的のスコープに応じて、クラスター管理者は **config.yaml** ファイルに新しいユーザーを追加できます。

```
...
scopes:
  - name: agent:create
    users: [<username_1>, <username_2>] 1
  - name: catalog:refresh
    users: [<username_3>, <username_4>]
  - name: config:refresh
    users: [<username_5>, <username_6>]
```

```
default:
  scopes:
    - rating:read
    - rating:write
...
```

1 Git リポジトリーホスティングサービスプロバイダーに登録されているユーザー名。



注記

初めてログインするユーザーは、**config.yaml** に追加されていても、デフォルトのスコープしかありません。追加のスコープをアクティブ化するには、ユーザーが少なくとも 1 回ログインしていることを確認してください。

2. **config.yaml** ファイルに **config-refresh** スコープがあることを確認してください。
3. 設定を更新します。

```
$ curl -X POST -H "Authorization: <access-token>" \ 1
    --header "Content-Type: application/json" \
    --data '{"force": true} \
    <api-route>/system/config/refresh
```

1 JWT トークン。

4.7.2. 開発者パースペクティブから Tekton Hub をオプトアウトする

クラスター管理者は、OpenShift Container Platform クラスターの **Developer** パースペクティブの **Pipeline builder** ページで、タスクやパイプラインなどの Tekton Hub リソースの表示をオプトアウトできます。

前提条件

- Red Hat OpenShift Pipelines Operator がクラスターにインストールされており、**oc** コマンドラインツールが使用可能であることを確認します。

手順

- **Developer** パースペクティブで Tekton Hub リソースを表示することを選択するには、**TektonConfig** カスタムリソース (CR) の **enable-devconsole-integration** フィールドの値を **false** に設定します。

```
apiVersion: operator.tekton.dev/v1alpha1
  kind: TektonConfig
  metadata:
    name: config
  spec:
    targetNamespace: openshift-pipelines
    ...
    hub:
      params:
```

```
      - name: enable-devconsole-integration
        value: "false"
...
```

デフォルトでは、**TektonConfig** CR には **enable-devconsole-integration** フィールドが含まれておらず、Red Hat OpenShift Pipelines Operator は値が **true** であると想定します。

4.7.3. 関連情報

- Tekton Hub の GitHub リポジトリー。
- OpenShift Pipelines のインストール
- Red Hat OpenShift Pipelines リリースノート

4.8. PIPELINES AS CODE の使用

Pipelines as Code を使用すると、クラスター管理者と必要な権限を持つユーザーは、パイプラインテンプレートをソースコード Git リポジトリーの一部として定義できます。設定された Git リポジトリーのソースコードプッシュまたはプルリクエストによってトリガーされると、この機能はパイプラインを実行し、ステータスを報告します。

4.8.1. 主な特長

Pipelines as Code は、次の機能をサポートしています。

- プルリクエストのステータスおよび Git リポジトリーをホストするプラットフォームの制御。
- GitHub は API を確認し、パイプライン実行のステータスを設定します (再チェックを含む)。
- GitHub のプルリクエストとコミットイベント。
- **/retest** などのコメントでリクエストアクションをプルします。
- Git イベントのフィルタリング、およびイベントごとの個別のパイプライン。
- ローカルタスク、Tekton Hub、およびリモート URL を含むパイプラインの自動タスク解決。
- GitHub blob およびオブジェクト API を使用した設定の取得。
- GitHub 組織を介して、または Prow スタイルの **OWNER** ファイルを使用したアクセス制御リスト (ACL)。
- ブートストラップおよび Pipelines as Code リポジトリーを管理するための **tkn pac** CLI プラグイン。
- GitHub App、GitHub Webhook、Bitbucket Server、および Bitbucket Cloud のサポート。

4.8.2. OpenShift Container Platform への Pipelines as Code のインストール

Pipelines as Code は、Red Hat OpenShift Pipelines Operator のインストール時にデフォルトでインストールされます。Pipelines 1.7 以降のバージョンを使用している場合は、Pipelines as Code を手動でインストールする手順を省略します。

Operator を使用して Pipelines as Code のデフォルトインストールを無効にするには、**TektonConfig** カスタムリソースで **enable** パラメーターの値を **false** に設定します。

```
...
 spec:
   platforms:
     openshift:
       pipelinesAsCode:
         enable: false
         settings:
           application-name: Pipelines as Code CI
           auto-configure-new-github-repo: "false"
           bitbucket-cloud-check-source-ip: "true"
           hub-catalog-name: tekton
           hub-url: https://api.hub.tekton.dev/v1
           remote-tasks: "true"
           secret-auto-create: "true"
...
```

必要に応じて、以下のコマンドを実行できます。

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": false}}}}}'
```

Red Hat OpenShift Pipelines Operator を使用して Pipelines as Code のデフォルトインストールを有効にするには、**TektonConfig** カスタムリソースで **enable** パラメーターの値を **true** に設定します。

```
...
spec:
   addon:
     enablePipelinesAsCode: false
...
```

必要に応じて、以下のコマンドを実行できます。

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": true}}}}}'
```

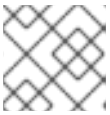
4.8.3. Pipelines as Code CLI のインストール

クラスター管理者は、ローカルマシンで、またはテスト用のコンテナーとして **tkn-pac** および **opc** CLI ツールを使用できます。**tkn pac** および **opc** CLI ツールは、Red Hat OpenShift Pipelines の **tkn** CLI をインストールすると自動的にインストールされます。

サポート対象プラットフォーム用の **tkn pac** および **opc** バージョン **1.9.1** バイナリーをインストールできます。

- Linux (x86_64, amd64)
- Linux on IBM Z and LinuxONE (s390x)
- Linux on IBM Power Systems (ppc64le)
- Mac

- Windows



注記

バイナリーは **tkn** バージョン **0.23.1** と互換性があります。

4.8.4. サービスプロバイダーをホストする Git リポジトリーでの Pipelines as Code の使用

Pipelines as Code をインストールした後に、クラスター管理者はサービスプロバイダーをホストする Git リポジトリーを設定できます。現在、以下のサービスがサポートされています。

- GitHub アプリケーション
- GitHub Webhook
- GitLab
- Bitbucket Server
- Bitbucket Cloud

注記

GitHub アプリケーションは、Pipelines as Code での使用に推奨されるサービスです。

4.8.5. GitHub アプリケーションでの Pipelines as Code の使用

GitHub アプリケーションは Red Hat OpenShift Pipeline とのインテグレーションポイントとして機能し、Git ベースのワークフローのメリットを OpenShift Pipelines にもたらします。クラスター管理者は、すべてのクラスターユーザーに単一の GitHub アプリケーションを設定できます。GitHub アプリケーションが Pipelines as Code と連携するには、GitHub アプリケーションの Webhook が GitHub イベントをリッスンする Pipelines as Code イベントリスナールート (または受信エンドポイント) をポイントするようにします。

4.8.5.1. GitHub アプリケーションの設定

クラスター管理者は、以下のコマンドを実行して GitHub アプリケーションを作成できます。

```
$ tkn pac bootstrap github-app
```

tkn pac CLI プラグインがインストールされていない場合は、GitHub アプリケーションを手動で作成できます。

手順

Pipelines as Code 用に GitHub アプリケーションを手動で作成および設定するには、以下の手順を実行します。

1. GitHub アカウントにサインインします。
2. **Settings → Developer settings → GitHub Apps** に移動し、**New GitHub App** をクリックします。
3. GitHub App フォームに以下の情報を入力します。

- GitHub Application Name: **OpenShift Pipelines**
- **Homepage URL**: OpenShift Console の URL
- **Webhook URL**: Pipelines as Code ルートまたは受信 URLこれは、コマンド **echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')** を実行して見つけることができます。
- **Webhook secret**: 任意のシークレット。コマンド **openssl rand -hex 20** を実行してシークレットを生成することができます。

4. 以下の **リポジトリーのパーミッション** を選択します。
   - **チェック**: **読み取り/書き込み**
   - **コンテンツ**: **読み取り/書き込み**
   - **問題**: **読み取り/書き込み**
   - **メタデータ**: **読み取り専用**
   - **プルリクエスト**: **読み取り/書き込み**
5. 以下の **組織のパーミッション** を選択します。
   - **メンバー**: **読み取り専用**
   - **プラン**: **読み取り専用**
6. 以下の **ユーザーパーミッション** を選択します。
   - **コミットコメント**
   - **問題のコメント**
   - **プルリクエスト**
   - **プルリクエストのレビュー**
   - **プルリクエストのレビューコメント**
   - **プッシュ**
7. **Create GitHub App**をクリックします。
8. 新たに作成された GitHub App の **Details** ページで、上部に表示される **App ID** を書き留めます。
9. **Private keys**セクションで、**Generate Private key**をクリックして GitHub アプリケーションの秘密鍵を自動的に生成およびダウンロードします。今後の参照や使用のために秘密鍵を安全に保管します。

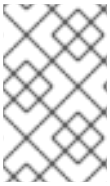
4.8.5.2. GitHub アプリケーションにアクセスするための Pipelines as Code の設定

新たに作成された GitHub アプリケーションにアクセスするために Pipelines as Code を設定するには、以下のコマンドを実行します。

+

```
$ oc -n openshift-pipelines create secret generic pipelines-as-code-secret \
    --from-literal github-private-key="$(cat <PATH_PRIVATE_KEY>)" \ 1
    --from-literal github-application-id="<APP_ID>" \ 2
    --from-literal webhook.secret="<WEBHOOK_SECRET>" 3
```

1. GitHub アプリケーションの設定時にダウンロードした秘密鍵へのパス。
2. GitHub アプリケーションの **App ID**。
3. GitHub アプリケーションの作成時に提供された Webhook シークレット。



注記

GitHub Enterprise から設定されたヘッダーを検出し、それを GitHub Enterprise API 承認 URL に使用することで、Pipelines as Code は自動的に GitHub Enterprise と連携します。

4.8.5.3. 管理者パースペクティブでの GitHub アプリケーションの作成

クラスター管理者は、OpenShift Container Platform クラスターで GitHub アプリケーションを Pipelines as Code を使用するように設定できます。この設定により、ビルドのデプロイに必要な一連のタスクを実行できます。

前提条件

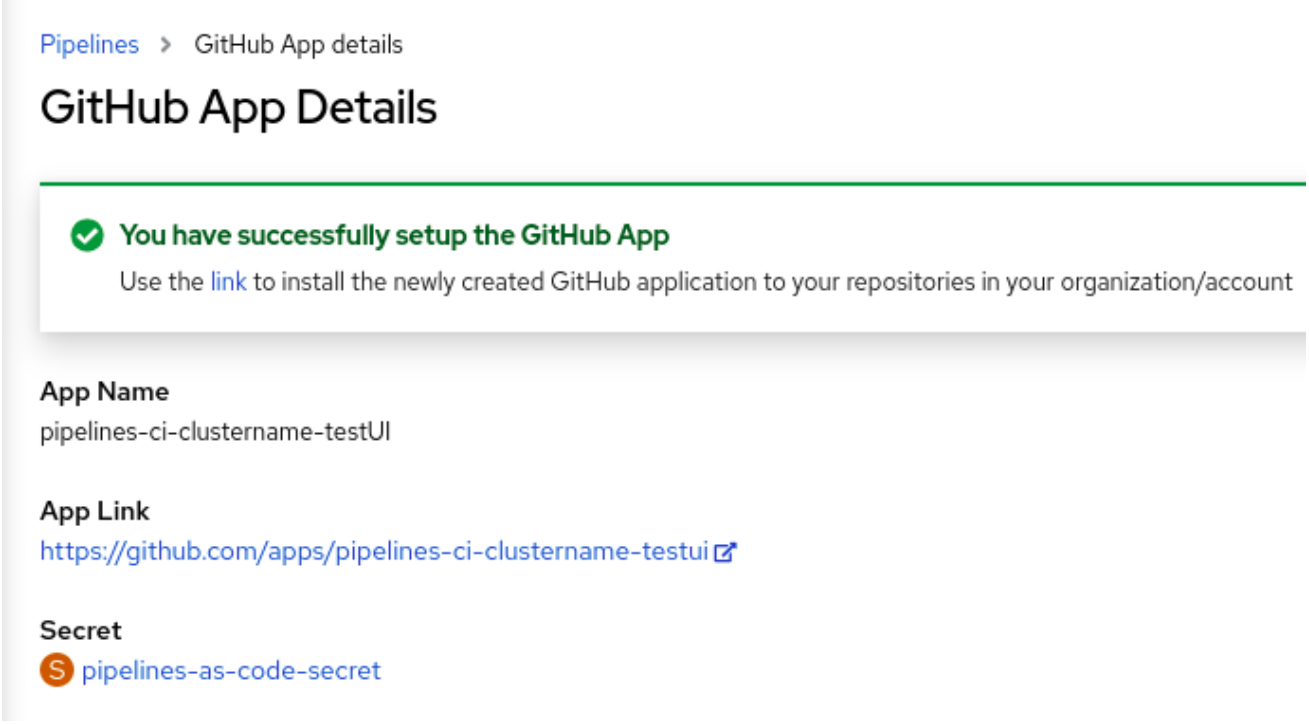
Operator Hub から Red Hat OpenShift Pipelines **pipelines-1.10** Operator をインストールしている。

手順

1. 管理者パースペクティブで、ナビゲーションペインを使用して **Pipelines** に移動します。
2. **Pipelines** ページで **GitHub アプリのセットアップ** をクリックします。
3. GitHub のアプリケーション名を入力します。例: **pipelines-ci-clustername-testui**
4. **Setup** をクリックします。
5. ブラウザーでプロンプトが表示されたら、Git パスワードを入力します。
6. **Create GitHub App for <username> をクリックします。ここで、<username>** は GitHub ユーザー名です。

検証

GitHub App の作成に成功すると、OpenShift Container Platform Web コンソールが開き、アプリケーションの詳細を表示します。



GitHub App の詳細は、**openShift-pipelines** namespace にシークレットとして保存されます。

GitHub アプリケーションに関連付けられている名前、リンク、シークレットなどの詳細を表示するには、**パイプライン** に移動し、**GitHub アプリの表示** をクリックします。

4.8.6. GitHub Webhook での Pipelines as Code の使用

GitHub アプリケーションを作成できない場合は、リポジトリーで GitHub Webhook で Pipelines as Code を使用します。ただし、GitHub Webhook で Pipelines as Code を使用しても、GitHub Check Runs API にはアクセスできません。タスクのステータスはプル要求のコメントとして追加され、**Checks** タブでは利用できません。

注記

GitHub Webhook を使用した Pipelines as Code は、**/retest** や **/ok-to-test** などの GitOps コメントには対応していません。継続的インテグレーション (CI) を再開するには、リポジトリーへの新しいコミットを作成します。たとえば、変更を加えずに新しいコミットを作成するには、次のコマンドを使用できます。

```
$ git --amend -a --no-edit && git push --force-with-lease <origin> <branchname>
```

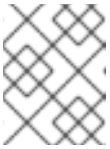
前提条件

- Pipelines as Code がクラスターにインストールされている。
- 承認用に GitHub で個人アクセストークンを作成する。
  - セキュアで粒度の細かいトークンを生成するには、そのスコープを特定のリポジトリーに制限し、以下のパーミッションを付与します。

    表4.7 粒度の細かいトークンのパーミッション

Name	アクセス
管理	Read-only
メタデータ	Read-only
コンテンツ	Read-only
コミットステータス	読み取りおよび書き込み
プルリクエスト	読み取りおよび書き込み
Webhook	読み取りおよび書き込み

- クラシックトークンを使用するには、パブリックリポジトリーのスコープを **public_repo** に設定し、プライベートリポジトリーのスコープを **repo** に設定します。さらに、トークンの有効期限を短くして、別の場所でトークンを書き留めておきます。



注記

tkn pac CLI を使用して Webhook を設定する必要がある場合は、**admin:repo_hook** スコープを追加します。

手順

1. Webhook を設定し、**リポジトリー** カスタムリソース (CR) を作成します。

- **tkn pac** CLI ツールを使用して webhook を設定し、**リポジトリー** CR を **自動的に** 作成するには、次のコマンドを使用します。

```
$ tkn pac create repo
```

対話型出力の例

```
? Enter the Git repository url (default: https://github.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository owner-repo has been created in repo-pipelines namespace
✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
sJNwdmTifHTs):  sJNwdmTifHTs
ℹ  You now need to create a GitHub personal access token, please checkout the docs at
https://docs.github.com/en/authentication/keeping-your-account-and-data-
secure/creating-a-personal-access-token for the required scopes
? Please enter the GitHub access token:  ****************************************
✓ Webhook has been created on repository owner/repo
  Webhook Secret owner-repo has been created in the repo-pipelines namespace.
```

```
 Repository CR owner-repo has been updated with webhook secret in the repo-pipelines namespace
ℹ Directory .tekton has been created.
✓ We have detected your repository using the programming language Go.
✓ A basic template has been created in /home/Go/src/github.com/owner/repo/.tekton/pipelinerun.yaml, feel free to customize it.
```

- Webhook を設定して **Repository** CR を **手動** で作成するには、以下の手順を実行します。

  i. OpenShift クラスターで、Pipelines as Code コントローラーの公開 URL を抽出します。

  ```
  $ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o jsonpath='{.spec.host}')
  ```

  ii. GitHub リポジトリーまたは組織で、以下の手順を実行します。

  A. **Settings** -> **Webhooks** に移動し、**Add webhook** をクリックします。

  B. **Payload URL** を Pipelines as Code コントローラーのパブリック URL に設定します。

  C. コンテンツタイプを **application/json** として選択します。

  D. Webhook シークレットを追加し、別の場所に書き留めます。**openssl** がローカルマシンにインストールされた状態で、ランダムなシークレットを生成します。

  ```
  $ openssl rand -hex 20
  ```

  E. **Let me select individual events** をクリックし、**Commit comments**、**Issue comments**、**Pull request**、および **Pushes** のイベントを選択します。

  F. **Add webhook** をクリックします。

  iii. OpenShift クラスターで、個人アクセストークンおよび Webhook シークレットを使用して **Secret** オブジェクトを作成します。

  ```
  $ oc -n target-namespace create secret generic github-webhook-config \
    --from-literal provider.token="<GITHUB_PERSONAL_ACCESS_TOKEN>" \
    --from-literal webhook.secret="<WEBHOOK_SECRET>"
  ```

  iv. **Repository** CR を作成します。

  例: Repository CR

  ```
  apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
  kind: Repository
  metadata:
    name: my-repo
    namespace: target-namespace
  spec:
    url: "https://github.com/owner/repo"
    git_provider:
      secret:
        name: "github-webhook-config"
  ```

```
    key: "provider.token" # Set this if you have a different key in your secret
  webhook_secret:
    name: "github-webhook-config"
    key: "webhook.secret" # Set this if you have a different key for your secret
```

注記

Pipelines as Code は、OpenShift **Secret** オブジェクトと **Repository** CR が同じ namespace にあることを前提としています。

2. オプション: 既存の **Repository** CR の場合、複数の GitHub Webhook シークレットを追加するか、削除されたシークレットの代わりを提供します。
   a. **tkn pac** CLI ツールを使用して Webhook を追加します。

   例: tkn pac CLI を使用した追加の Webhook

   ```
   $ tkn pac webhook add -n repo-pipelines
   ```

   対話型出力の例

   ```
   ✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
    I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
   ? Do you want me to use it? Yes
   ? Please enter the secret to configure the webhook for payload validation (default: AeHdHTJVfAeH):  AeHdHTJVfAeH
   ✓ Webhook has been created on repository owner/repo
    Secret owner-repo has been updated with webhook secert in the repo-pipelines namespace.
   ```

   b. 既存の OpenShift **Secret** オブジェクトの **webhook.secret** キーを更新します。

3. オプション: 既存の **Repository** CR の場合は、パーソナルアクセストークンを更新します。
   - **tkn pac** CLI ツールを使用してパーソナルアクセストークンを更新します。

     例: tkn pac CLI を使用したパーソナルアクセストークンの更新

     ```
     $ tkn pac webhook update-token -n repo-pipelines
     ```

     対話型出力の例

     ```
     ? Please enter your personal access token:  ****************************************
      Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
     ```

   - または、**Repository** CR を変更してパーソナルアクセストークンを更新します。
     i. **Repository** CR でシークレットの名前を見つけます。

     ```
     ...
     spec:
     ```

```
  git_provider:
    secret:
      name: "github-webhook-config"
...
```

ii. **oc patch** コマンドを使用して、**$target_namespace** namespace の **$NEW_TOKEN** の値を更新します。

```
$ oc -n $target_namespace patch secret github-webhook-config -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

関連情報

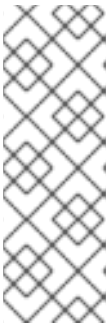
- GitHub Webhook documentation on GitHub
- GitHub Check Runs documentation on GitHub
- Creating a personal access token on GitHub
- Classic tokens with pre-filled permissions

4.8.7. GitLab での Pipelines as Code の使用

組織またはプロジェクトが優先プラットフォームとして GitLab を使用する場合は、GitLab の Webhook を使用してリポジトリーの Pipelines as Code を使用できます。

前提条件

- Pipelines as Code がクラスターにインストールされている。
- 承認には、GitLab のプロジェクトまたは組織のマネージャーとしてパーソナルアクセストークンを生成します。



注記

- **tkn pac** CLI を使用して Webhook を設定する必要がある場合は、**admin:repo_hook** スコープをトークンに追加します。
- 特定のプロジェクトを対象とするトークンを使用しても、フォークされたリポジトリーから送信されたマージリクエスト (MR) に API でのアクセスはできません。このような場合、Pipelines as Code はパイプラインの結果を MR のコメントとして表示します。

手順

1. Webhook を設定し、**リポジトリー** カスタムリソース (CR) を作成します。
   - **tkn pac** CLI ツールを使用して webhook を設定し、**リポジトリー** CR を **自動的に** 作成するには、次のコマンドを使用します。

```
$ tkn pac create repo
```

対話型出力の例

```
? Enter the Git repository url (default: https://gitlab.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository repositories-project has been created in repo-pipelines namespace
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
? Please enter the project ID for the repository you want to be configured,
  project ID refers to an unique ID (e.g. 34405323) shown at the top of your GitLab project
: 17103
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
lFjHIEcaGFlF):  lFjHIEcaGFlF
ℹ  You now need to create a GitLab personal access token with `api` scope
ℹ  Go to this URL to generate one https://gitlab.com/-/profile/personal_access_tokens,
see https://is.gd/rOEo9B for documentation
? Please enter the GitLab access token:  **************************
? Please enter your GitLab API URL::  https://gitlab.com
✓ Webhook has been created on your repository
  Webhook Secret repositories-project has been created in the repo-pipelines
namespace.
  Repository CR repositories-project has been updated with webhook secret in the repo-
pipelines namespace
ℹ Directory .tekton has been created.
✓ A basic template has been created in
/home/Go/src/gitlab.com/repositories/project/.tekton/pipelinerun.yaml, feel free to
customize it.
```

- Webhook を設定して **Repository** CR を **手動** で作成するには、以下の手順を実行します。

  i. OpenShift クラスターで、Pipelines as Code コントローラーの公開 URL を抽出します。

  ```
  $ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
  jsonpath='{.spec.host}')
  ```

  ii. GitLab プロジェクトで、以下の手順を実行します。

  A. 左側のサイドバーを使用して **Settings** -> **Webhooks** に移動します。

  B. **URL** を Pipelines as Code コントローラーのパブリック URL に設定します。

  C. Webhook シークレットを追加し、別の場所に書き留めます。**openssl** がローカルマシンにインストールされた状態で、ランダムなシークレットを生成します。

  ```
  $ openssl rand -hex 20
  ```

  D. **Let me select individual events** をクリックし、**Commit comments**、**Issue comments**、**Pull request**、および **Pushes** のイベントを選択します。

  E. **Save Changes** をクリックします。

  iii. OpenShift クラスターで、個人アクセストークンおよび Webhook シークレットを使用して **Secret** オブジェクトを作成します。

```
$ oc -n target-namespace create secret generic gitlab-webhook-config \
  --from-literal provider.token="<GITLAB_PERSONAL_ACCESS_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. **Repository** CR を作成します。

例: Repository CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://gitlab.com/owner/repo" 1
  git_provider:
    secret:
      name: "gitlab-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "gitlab-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

1 現時点で、Pipelines as Code では GitLab のプライベートインスタンスは自動検出されません。このような場合には、**git_provider.url** 仕様の下に API URL を指定します。通常、**git_provider.url** 仕様を使用して API URL を手動で上書きできます。

注記

- Pipelines as Code は、OpenShift **Secret** オブジェクトと **Repository** CR が同じ namespace にあることを前提としています。

2. オプション: 既存の **Repository** CR の場合、複数の GitLab Webhook シークレットを追加するか、削除されたシークレットの代わりを提供します。

a. **tkn pac** CLI ツールを使用して Webhook を追加します。

例: tkn pac CLI を使用した Webhook の追加

```
$ tkn pac webhook add -n repo-pipelines
```

対話型出力の例

```
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
AeHdHTJVfAeH):  AeHdHTJVfAeH
```

```
✓ Webhook has been created on repository owner/repo
 Secret owner-repo has been updated with webhook secert in the repo-pipelines
namespace.
```

b. 既存の OpenShift **Secret** オブジェクトの **webhook.secret** キーを更新します。

3. オプション: 既存の **Repository** CR の場合は、パーソナルアクセストークンを更新します。

- **tkn pac** CLI ツールを使用してパーソナルアクセストークンを更新します。

  例: tkn pac CLI を使用したパーソナルアクセストークンの更新

```
$ tkn pac webhook update-token -n repo-pipelines
```

  対話型出力の例

```
? Please enter your personal access token:  ****************************************
 Secret owner-repo has been updated with new personal access token in the repo-
pipelines namespace.
```

- または、**Repository** CR を変更してパーソナルアクセストークンを更新します。

  i. **Repository** CR でシークレットの名前を見つけます。

```
...
spec:
  git_provider:
    secret:
      name: "gitlab-webhook-config"
...
```

  ii. **oc patch** コマンドを使用して、**$target_namespace** namespace の **$NEW_TOKEN** の値を更新します。

```
$ oc -n $target_namespace patch secret gitlab-webhook-config -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

関連情報

- GitLab Webhook documentation on GitLab

4.8.8. Bitbucket Cloud での Pipelines as Code の使用

組織またはプロジェクトが優先プラットフォームとして Bitbucket Cloud を使用する場合、Bitbucket Cloud の Webhook を使用してリポジトリーに Pipelines as Code を使用できます。

前提条件

- Pipelines as Code がクラスターにインストールされている。
- Bitbucket Cloud でアプリのパスワードを作成する。
  - 以下のボックスをチェックして、適切なパーミッションをトークンに追加します。

- アカウント: **メール**、**読み取り**
- ワークスペースのメンバーシップ: **読み取り**、**書き込み**
- プロジェクト: **読み取り**、**書き込み**
- 問題: **読み取り**、**書き込み**
- プルリクエスト: **読み取り**、**書き込み**

注記

- **tkn pac** CLI を使用して Webhook を設定する必要がある場合は、**Webhooks**:**Read** と **Write** パーミッションをトークンに追加します。
- 生成されたら、パスワードまたはトークンのコピーを別の場所に保存します。

手順

1. Webhook を設定し、**Repository** CR を作成します。

- **tkn pac** CLI ツールを使用して webhook を設定し、**リポジトリー** CR を **自動的に** 作成するには、次のコマンドを使用します。

```
$ tkn pac create repo
```

対話型出力の例

```
? Enter the Git repository url (default: https://bitbucket.org/workspace/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository workspace-repo has been created in repo-pipelines namespace
✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
? Please enter your bitbucket cloud username:  <username>
ℹ  You now need to create a Bitbucket Cloud app password, please checkout the docs at https://is.gd/fqMHiJ for the required permissions
? Please enter the Bitbucket Cloud app password:  ************************************
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
✓ Webhook has been created on repository workspace/repo
  Webhook Secret workspace-repo has been created in the repo-pipelines namespace.
  Repository CR workspace-repo has been updated with webhook secret in the repo-pipelines namespace
ℹ Directory .tekton has been created.
✓ A basic template has been created in /home/Go/src/bitbucket/repo/.tekton/pipelinerun.yaml, feel free to customize it.
```

- Webhook を設定して **Repository** CR を **手動** で作成するには、以下の手順を実行します。

i. OpenShift クラスターで、Pipelines as Code コントローラーの公開 URL を抽出します。

```
$ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
jsonpath='{.spec.host}')
```

ii. Bitbucket Cloud で、以下の手順を実行します。

A. Bitbucket Cloud リポジトリーの左側のナビゲーションペインを使用して **Repository settings** -> **Webhooks** に移動し、**Add webhook** をクリックします。

B. **Title** を設定します。たとえば、Pipelines as Code です。

C. **URL** を Pipelines as Code コントローラーのパブリック URL に設定します。

D. **Repository: Push**、**Pull Request: Created**、**Pull Request: Updated**、および **Pull Request: Comment created** のイベントを選択します。

E. **Save** をクリックします。

iii. OpenShift クラスターで、ターゲット namespace に app パスワードを使用して **Secret** オブジェクトを作成します。

```
$ oc -n target-namespace create secret generic bitbucket-cloud-token \
  --from-literal provider.token="<BITBUCKET_APP_PASSWORD>"
```

iv. **Repository** CR を作成します。

例: Repository CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  branch: "main"
  git_provider:
    user: "<BITBUCKET_USERNAME>" (1)
    secret:
      name: "bitbucket-cloud-token" (2)
      key: "provider.token" # Set this if you have a different key in your secret
```

(1) 所有者ファイルの **ACCOUNT_ID** からしかユーザーの参照はできません。

(2) Pipelines as Code は、**git_provider.secret** 仕様で参照され、**Repository** CR が同じ namespace にあることを前提としています。

注記

- **tkn pac create** および **tkn pac bootstrap** コマンドは Bitbucket Cloud ではサポートされていません。
- Bitbucket Cloud では Webhook シークレットはサポートされません。ペイロードを保護し、CI のハイジャックを防止するために、Pipelines as Code は Bitbucket Cloud IP アドレスのリストをフェッチし、Webhook の受信がそれらの IP アドレスからのみ行われるようにします。
  - デフォルトの動作を無効にするには、**pipelines-as-code** namespace の Pipelines as Code config map で **bitbucket-cloud-check-source-ip キー**を **false** に設定します。
  - 追加の安全な IP アドレスまたはネットワークを許可するには、**pipelines-as-code** namespace の Pipelines as Code config map の **bitbucket-cloud-additional-source-ip** キーにコンマ区切りの値として追加します。

2. オプション: 既存の **Repository** CR の場合は、複数の Bitbucket Cloud Webhook シークレットを追加するか、削除されたシークレットの代わりに指定します。

   a. **tkn pac** CLI ツールを使用して Webhook を追加します。

   例: tkn pac CLI を使用した Webhook の追加

   ```
   $ tkn pac webhook add -n repo-pipelines
   ```

   対話型出力の例

   ```
   ✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
   ? Please enter your bitbucket cloud username:  <username>
     I have detected a controller url: https://pipelines-as-code-controller-openshift-
   pipelines.apps.example.com
   ? Do you want me to use it? Yes
   ✓ Webhook has been created on repository workspace/repo
     Secret workspace-repo has been updated with webhook secret in the repo-pipelines
   namespace.
   ```

   注記

   -n <namespace> オプションを **tkn pac webhook add** コマンドで使用するのは、**Repository** CR がデフォルト以外の namespace に存在する場合のみです。

   b. 既存の OpenShift **Secret** オブジェクトの **webhook.secret** キーを更新します。

3. オプション: 既存の **Repository** CR の場合は、パーソナルアクセストークンを更新します。

   - **tkn pac** CLI ツールを使用してパーソナルアクセストークンを更新します。

   例: tkn pac CLI を使用したパーソナルアクセストークンの更新

   ```
   $ tkn pac webhook update-token -n repo-pipelines
   ```

対話型出力の例

```
? Please enter your personal access token: ****************************************
  Secret owner-repo has been updated with new personal access token in the repo-
pipelines namespace.
```

注記

-n <namespace> オプションを **tkn pac webhook update-token** コマンドで使用するのは、**Repository** CR がデフォルト以外の namespace に存在する場合のみです。

- または、**Repository** CR を変更してパーソナルアクセストークンを更新します。
  i. **Repository** CR でシークレットの名前を見つけます。

```
...
spec:
  git_provider:
    user: "<BITBUCKET_USERNAME>"
    secret:
      name: "bitbucket-cloud-token"
      key: "provider.token"
...
```

  ii. **oc patch** コマンドを使用して、**$target_namespace** namespace の **$password** の値を更新します。

```
$ oc -n $target_namespace patch secret bitbucket-cloud-token -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

関連情報

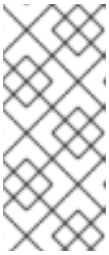
- Creating app password on Bitbucket Cloud
- Introducing Altassian Account ID and Nicknames

4.8.9. Bitbucket サーバーでの Pipelines as Code の使用

組織またはプロジェクトが優先プラットフォームとして Bitbucket Server を使用する場合は、Bitbucket Server の Webhook でリポジトリーの Pipelines as Code を使用できます。

前提条件

- Pipelines as Code がクラスターにインストールされている。
- Bitbucket Server でプロジェクトのマネージャーとしてパーソナルアクセストークンを生成し、そのコピーを別の場所に保存します。



注記

- トークンには、**PROJECT_ADMIN** および **REPOSITORY_ADMIN** 権限が必要です。
- トークンには、プルリクエストでフォークされたリポジトリーへのアクセスが必要です。

手順

1. OpenShift クラスターで、Pipelines as Code コントローラーの公開 URL を抽出します。

```
$ echo https://$(oc get route -n pipelines-as-code pipelines-as-code-controller -o
jsonpath='{.spec.host}')
```

2. Bitbucket Server で、以下の手順を実行します。

   a. Bitbucket Data Center リポジトリーの左側のナビゲーションペインを使用して **Repository settings** -> **Webhooks** に移動し、**Add webhook** をクリックします。

   b. **Title** を設定します。たとえば、Pipelines as Code です。

   c. **URL** を Pipelines as Code コントローラーのパブリック URL に設定します。

   d. Webhook シークレットを追加し、そのコピーを別の場所に保存します。**openssl** をローカルマシンにインストールしている場合は、以下のコマンドを使用してランダムなシークレットを生成します。

```
$ openssl rand -hex 20
```

   e. 以下のイベントを選択します。

      - **Repository: Push**
      - **Repository: Modified**
      - **Pull Request: Opened**
      - **Pull Request: Source branch updated**
      - **Pull Request: Comment added**

   f. **Save** をクリックします。

3. OpenShift クラスターで、ターゲット namespace に app パスワードを使用して **Secret** オブジェクトを作成します。

```
$ oc -n target-namespace create secret generic bitbucket-server-webhook-config \
  --from-literal provider.token="<PERSONAL_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

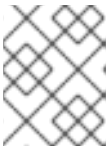
4. **Repository** CR を作成します。

   例: Repository CR

```
---
```

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  git_provider:
    url: "https://bitbucket.server.api.url/rest" (1)
    user: "<BITBUCKET_USERNAME>" (2)
    secret: (3)
      name: "bitbucket-server-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "bitbucket-server-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

1. **/api/v1.0** 接尾辞のない正しい Bitbucket Server API URL があることを確認します。通常、デフォルトのインストールには **/rest** 接尾辞があります。
2. 所有者ファイルの **ACCOUNT_ID** からしかユーザーの参照はできません。
3. Pipelines as Code は、**git_provider.secret** 仕様で参照され、**Repository** CR が同じ namespace にあることを前提としています。



注記

tkn pac create および **tkn pac bootstrap** コマンドは Bitbucket サーバーではサポートされません。

関連情報

- Creating personal tokens on Bitbucket Server
- Creating webhooks on Bitbucket server

4.8.10. Pipelines as Code とカスタム証明書のインターフェイス

プライベートに署名またはカスタム証明書を使用してアクセス可能な Git リポジトリーで Pipelines as Code を設定するには、証明書を Pipelines as Code に公開できます。

手順

- Red Hat OpenShift Pipelines Operator を使用して Pipelines as Code をインストールしている場合、**Proxy** オブジェクトを使用してカスタム証明書をクラスターに追加できます。Operator は、Pipelines as Code を含むすべての Red Hat OpenShift Pipelines コンポーネントおよびワークロードの証明書を公開します。

関連情報

- クラスター全体のプロキシーの有効化

4.8.11. Pipelines as Code での Repository CRD の使用

Repository カスタムリソース (CR) には、次の主要な機能があります。

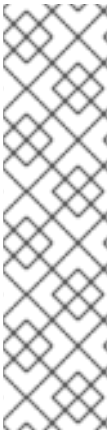
- URL からのイベントの処理について Pipelines as Code に通知します。
- Pipeline 実行の namespace について Pipelines as Code に通知します。
- Webhook メソッドを使用する場合、Git プロバイダープラットフォームに必要な API シークレット、ユーザー名、または API URL を参照します。
- リポジトリーの最後のパイプライン実行ステータスを指定します。

tkn pac CLI またはその他の代替方法を使用して、ターゲット namespace 内に **Repository** CR を作成できます。以下に例を示します。

```
cat <<EOF|kubectl create -n my-pipeline-ci -f- 1

apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: project-repository
spec:
  url: "https://github.com/<repository>/<project>"
EOF
```

1 **my-pipeline-ci** はターゲット namespace です。

https://github.com/<repository>/<project> などの URL からイベントが発生すると、Pipelines as Code はその URL とマッチさせ、**<repository>/<project>** リポジトリーのコンテンツのチェックアウトを開始し、パイプラインを実行して **.tekton/** ディレクトリーのコンテンツとマッチさせます。



注記

- ソースコードリポジトリーに関連付けられたパイプラインが実行されるのと同じ namespace に **Repository** CRD を作成する必要があります。これは別の namespace をターゲットにすることはできません。
- 複数の **リポジトリー** CRD が同じイベントとマッチする場合には、Pipelines as Code は最も古いもののみを処理します。特定の namespace と同じにする必要がある場合は、**pipelinesascode.tekton.dev/target-namespace: "<mynamespace>"** アノテーションを追加します。このような明示的なターゲティングにより、悪意のあるアクターがアクセス権のない namespace でパイプラインの実行を防ぎます。

4.8.11.1. Repository CRD での同時実行制限の設定

Repository CRD の **concurrency_limit** 仕様を使用して、リポジトリーに対して同時に実行されるパイプライン実行の最大数を定義できます。

```
...
spec:
  concurrency_limit: <number>
  ...
```

イベントに一致する複数のパイプラインが実行される場合、パイプラインは、イベントの開始に一致するアルファベット順に実行されます。

たとえば、**.tekton** ディレクトリーに 3 つのパイプラインが実行され、リポジトリー設定に **concurrency_limit** が **1** のプルリクエストを作成する場合、すべてのパイプライン実行はアルファベット順に実行されます。常に 1 つのパイプライン実行のみが running 状態にあり、残りはキューに入れられます。

4.8.12. Pipelines as Code リゾルバーの使用

Pipelines as Code リゾルバーは、実行中のパイプライン実行が他のパイプライン実行と競合しないようにします。

パイプラインとパイプライン実行を分割するには、ファイルを **.tekton/** ディレクトリーまたはそのサブディレクトリーに保存します。

Pipelines as Code が、**.tekton/** ディレクトリーにある YAML ファイル内のタスクまたはパイプラインへの参照を使用してパイプライン実行を監視すると、Pipelines as Code は、参照されたタスクを自動的に解決して、**PipelineRun** オブジェクトに埋め込まれた仕様と合わせて単一のパイプラインを実行します。

Pipelines as Code が **Pipeline** または **PipelineSpec** 定義で参照されるタスクを解決できない場合に、実行はクラスターに適用される前に失敗します。Git プロバイダープラットフォームと、**Repository** CR が置かれているターゲット namespace のイベント内で問題を確認できます。

リゾルバーは、以下のタイプのタスクを監視する場合に解決を省略します。

- クラスタータスクへの参照。
- タスクまたはパイプラインバンドル。
- API バージョンに **tekton.dev/** 接頭辞のないカスタムタスク。

リゾルバーは、そのようなタスクを変換せずにそのまま使用します。

プルリクエストに送信する前にパイプライン実行をローカルでテストするには、**tkn pac resolve** コマンドを使用します。

リモートパイプラインおよびタスクを参照することもできます。

4.8.12.1. Pipelines as Code でのリモートタスクアノテーションの使用

Pipelines as Code は、パイプライン実行でアノテーションを使用してリモートタスクまたはパイプラインの取得をサポートします。パイプライン実行、または **PipelineRun** または **PipelineSpec** オブジェクトのパイプラインでリモートタスクを参照する場合に、Pipelines as Code リゾルバーにはこれが自動的に含まれます。リモートタスクのフェッチまたは解析中にエラーが発生した場合、Pipelines as Code はタスクの処理を停止します。

リモートタスクを含めるには、以下のアノテーションの例を参照してください。

Tekton Hub でのリモートタスクの参照

- Tekton Hub で単一のリモートタスクを参照します。

```
...
  pipelinesascode.tekton.dev/task: "git-clone" (1)
...
```

(1) Pipelines as Code には、Tekton Hub からのタスクの最新バージョンが含まれています。

- Tekton Hub から複数のリモートタスクを参照します。

```
...
  pipelinesascode.tekton.dev/task: "[git-clone, golang-test, tkn]"
...
```

- **-<NUMBER>** 接尾辞を使用して、Tekton Hub から複数のリモートタスクを参照します。

```
...
  pipelinesascode.tekton.dev/task: "git-clone"
  pipelinesascode.tekton.dev/task-1: "golang-test"
  pipelinesascode.tekton.dev/task-2: "tkn" (1)
...
```

(1) デフォルトでは、Pipelines as Code は文字列を Tekton Hub から取得する最新のタスクとして解釈します。

- Tekton Hub からリモートタスクの特定のバージョンを参照します。

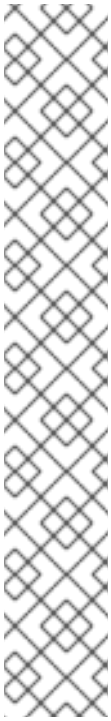
```
...
  pipelinesascode.tekton.dev/task: "[git-clone:0.1]" (1)
...
```

(1) Tekton Hub からの **git-clone** リモートタスクの **0.1** バージョンを参照します。

URL を使用するリモートタスク

```
...
  pipelinesascode.tekton.dev/task: "<https://remote.url/task.yaml>" (1)
...
```

(1) リモートタスクへの公開 URL。



注記

- GitHub とリモートタスクの URL を使用して **Repository** CRD と同じホストを使用する場合、Pipelines as Code は GitHub トークンを使用し、GitHub API を使用して URL を取得します。
  たとえば、**https://github.com/<organization>/<repository>** のようなリポジトリー URL があり、リモート HTTP URL が **https://github.com/<organization>/<repository>/blob/<mainbranch>/<path>/<file>** のような GitHub ブロブを参照している場合に、Pipelines as Code は、GitHub アプリトークンを使用して、そのプライベートリポジトリーからタスク定義ファイルをフェッチします。

  パブリック GitHub リポジトリーで作業する場合、Pipelines as Code は **https://raw.githubusercontent.com/<organization>/<repository>/<mainbranch>/<path>/<file>** などの GitHub の raw URL と同様に機能します。

- GitHub アプリケーショントークンは、リポジトリーが置かれている所有者または組織に対してスコープが設定されます。GitHub Webhook メソッドを使用すると、個人トークンが許可されている任意の組織のプライベートまたはパブリックリポジトリーを取得できます。

リポジトリー内の YAML ファイルからのタスク参照

```
...
pipelinesascode.tekton.dev/task: "<share/tasks/git-clone.yaml>" (1)
...
```

(1) タスク定義を含むローカルファイルへの相対パス。

4.8.12.2. Pipelines as Code でのリモートパイプラインアノテーションの使用

リモートパイプラインアノテーションを使用すると、複数のリポジトリーでパイプライン定義を共有できます。

```
...
  pipelinesascode.tekton.dev/pipeline: "<https://git.provider/raw/pipeline.yaml>" (1)
...
```

(1) リモートパイプライン定義への URL。同じリポジトリー内のファイルの場所を指定することもできます。

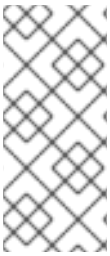
注記

アノテーションを使用してパイプライン定義を 1 つだけ参照できます。

4.8.13. Pipelines as Code を使用したパイプライン実行の作成

Pipelines as Code を使用してパイプラインを実行するには、リポジトリーの **.tekton/** ディレクトリーにパイプライン定義またはテンプレートを YAML ファイルとして作成します。リモート URL を使用して他のリポジトリー内の YAML ファイルを参照できますが、パイプラインの実行は、**.tekton/** ディレクトリーを含むリポジトリー内のイベントによってのみトリガーされます。

Pipelines as Code リゾルバーは、パイプラインの実行をすべてのタスクと共に、外部依存関係のない単一のパイプラインの実行としてバンドルします。



注記

- Pipeline の場合、spec または分離された **Pipeline** オブジェクトと共に少なくとも 1 つのパイプライン実行を使用します。
- タスクの場合、パイプライン内にタスク仕様を埋め込むか、Task オブジェクトとして個別に定義します。

コミットと URL のパラメーター化

{{<var>}} 形式の動的でデプロイメント可能な変数を使用して、コミットと URL のパラメーターを指定できます。現在、以下の変数を使用できます。

- **{{repo_owner}}**: リポジトリーの所有者。
- **{{repo_name}}**: リポジトリー名。
- **{{repo_url}}**: リポジトリーの完全な URL。
- **{{revision}}**: コミットの完全 SHA リビジョン。
- **{{sender}}**: コミットの送信者のユーザー名またはアカウント ID。
- **{{source_branch}}**: イベントが発生したブランチ名。
- **{{target_branch}}**: イベントが対象とするブランチ名。プッシュイベントの場合、これは **source_branch** と同じです。
- **{{pull_request_number}}**: **pull_request** イベントタイプに対してのみ定義されたプルまたはマージリクエスト番号。
- **{{git_auth_secret}}**: プライベートリポジトリーをチェックアウトするための Git プロバイダーのトークンで自動的に生成されるシークレット名。

イベントのパイプライン実行へのマッチング

パイプライン実行の特別なアノテーションを使用して、異なる Git プロバイダーイベントを各パイプラインに一致させることができます。イベントトガッチする複数のパイプライン実行がある場合に、Pipelines as Code はそれらを並行して実行し、パイプライン実行の終了直後に結果を Git プロバイダーに Post します。

プルイベントのパイプライン実行へのマッチング

次の例を使用して、**main** ブランチを対象とする **pull_request** イベントと、**pipeline-pr-main** パイプラインをマッチさせることができます。

```
...
  metadata:
    name: pipeline-pr-main
  annotations:
    pipelinesascode.tekton.dev/on-target-branch: "[main]" 1
    pipelinesascode.tekton.dev/on-event: "[pull_request]"
...
```

1 コンマ区切りのエントリーを追加して、複数のブランチを指定できます。たとえば、**"[main, release-nightly]"** です。さらに、以下を指定できます。

- **refs/heads/main** などのブランチへの完全な参照
- **refs/heads/*** などのパターンマッチングを含む glob
- **refs/tags/1.*** などのタグ

プッシュイベントのパイプライン実行とのマッチング

次の例を使用して、**pipeline-push-on-main** パイプラインを **refs/heads/main** ブランチを対象とする **プッシュ** イベントとマッチさせることができます。

```
...
  metadata:
    name: pipeline-push-on-main
  annotations:
    pipelinesascode.tekton.dev/on-target-branch: "[refs/heads/main]" 1
    pipelinesascode.tekton.dev/on-event: "[push]"
...
```

1 コンマ区切りのエントリーを追加することで、複数のブランチを指定できます。たとえば、**"[main, release-nightly]"** です。さらに、以下を指定できます。

- **refs/heads/main** などのブランチへの完全な参照
- **refs/heads/*** などのパターンマッチングを含む glob
- **refs/tags/1.*** などのタグ

高度なイベントマッチング

コードとしてのパイプラインは、高度なイベントマッチングのための Common Expression Language (CEL) ベースのフィルタリングの使用をサポートします。パイプラインの実行に **pipelinesascode.tekton.dev/on-cel-expression** アノテーションがある場合に、Pipelines as Code は CEL 式を使用し、**on-target-branch** アノテーションをスキップします。単純な **オンターゲットブランチ** アノテーションマッチングと比較して、CEL 式では複雑なフィルタリングと否定が可能です。

Pipelines as Code で CEL ベースのフィルタリングを使用するには、次のアノテーションの例を検討してください。

- **main** ブランチを対象とし、**wip** ブランチからの **pull_request** イベントを一致させるには、次のようにします。

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch == "main" && source_branch == "wip"
...
```

- パスが変更された場合にのみパイプラインを実行するには、glob パターンで **.pathChanged** 接尾辞関数を使用できます。

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
```

```
    event == "pull_request" && "docs/\*.md".pathChanged() (1)
...
```

(1) **docs** ディレクトリー内のすべてのマークダウンファイルと一致します。

- **[DOWNSTREAM]** で始まるすべてのプルリクエストとマッチさせるには、以下を実行します。

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request && event_title.startsWith("[DOWNSTREAM]")
...
```

- **pull_request** イベントでパイプラインを実行し、**experimental** ブランチを省略するには、以下を実行します。

```
...
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch != experimental"
...
```

Pipelines as Code を使用しながら高度な CEL ベースのフィルタリングを行うには、次のフィールドと接尾辞関数を使用できます。

- **event**: **push** または **pull_request** イベント。
- **target_branch**: ターゲットブランチ。
- **source_branch**: 元の **pull_request** イベントのブランチ。**push** イベントの場合は、**target_branch** と同じです。
- **event_title**: **push** イベントのコミットタイトルや、**pull_request** イベントのプルまたはマージリクエストのタイトルなど、イベントのタイトルとマッチします。現在、サポートされているプロバイダーは GitHub、Gitlab、および Bitbucket Cloud のみです。
- **.pathChanged**: 文字列への接尾辞関数です。文字列は、パスが変更されたかどうかを確認するパスの glob にすることができます。現在、GitHub と Gitlab のみがプロバイダーとしてサポートされています。

Github API 操作への一時的な GitHub App トークンの使用

GitHub API にアクセスするための Pipelines as Code によって生成された一時的なインストールトークンを使用できます。トークン値は **git-provider-token** キーのプライベートリポジトリー用に生成された一時的な **{{git_auth_secret}}** 動的変数に格納されます。

たとえば、プル要求にコメントを追加するには、Pipelines as Code アノテーションを使用して Tekton Hub からの **github-add-comment** タスクを使用できます。

```
...
  pipelinesascode.tekton.dev/task: "github-add-comment"
...
```

その後、タスクをパイプライン実行定義の **tasks** セクションまたは **finally** タスクに追加できます。

```
[...]
```

```
tasks:
  - name:
      taskRef:
        name: github-add-comment
      params:
        - name: REQUEST_URL
          value: "{{ repo_url }}/pull/{{ pull_request_number }}" (1)
        - name: COMMENT_OR_FILE
          value: "Pipelines as Code IS GREAT!"
        - name: GITHUB_TOKEN_SECRET_NAME
          value: "{{ git_auth_secret }}"
        - name: GITHUB_TOKEN_SECRET_KEY
          value: "git-provider-token"
...
```

(1) 動的変数を使用すると、任意のリポジトリーからのプルリクエストに対して、このスニペットテンプレートを再利用できます。

注記

GitHub アプリでは、生成されたインストールトークンは 8 時間利用可能で、クラスターで別の設定を行わない限り、イベントの発生元のリポジトリーにスコープが設定されます。

関連情報

- CEL language specification

4.8.14. Pipelines as Code を使用したパイプライン実行

デフォルト設定では、Pipelines as Code は、プルリクエストやプッシュなどの指定されたイベントがリポジトリーで発生したときに、リポジトリーのデフォルトブランチの **.tekton/** ディレクトリーで実行されるすべてのパイプラインを実行します。たとえば、デフォルトのブランチで実行されるパイプラインに、アノテーション **pipelinesascode.tekton.dev/on-event: "[pull_request]"** がある場合に、これはプル要求イベントが発生するたびに実行されます。

プルリクエストまたはマージリクエストが発生した場合、プルリクエストの作成者が次の条件を満たしている場合、Pipelines as Code はデフォルトブランチ以外のブランチからもパイプラインを実行します。

- 作成者はリポジトリーの所有者です。
- 作成者は、リポジトリーのコラボレーターです。
- 作成者はリポジトリーの組織のパブリックメンバーです。
- プルリクエストの作成者は、リポジトリーの GitHub 設定で定義されているように、**main** ブランチのリポジトリールートにある **OWNER** ファイルに一覧表示されます。また、プルリクエストの作成者は、**approvers** または **reviewers** セクションに追加されます。たとえば、作成者が **approvers** セクションにリストされている場合、その作成者が発行したプルリクエストによってパイプラインの実行が開始されます。

```
...
  approvers:
```

```
    - approved
...
```

プル要求の作成者は、要件を満たす別のユーザーがプル要求で **/ok-to-test** をコメントして、パイプライン実行を開始できます。

パイプライン実行

パイプラインの実行は常に、イベントを生成したリポジトリーに関連付けられた **Repository** CRD の namespace で実行されます。

tkn pac CLI ツールを使用して、パイプライン実行を確認できます。

- 最後のパイプライン実行を追跡するには、以下の例を使用します。

```
$ tkn pac logs -n <my-pipeline-ci> -L 1
```

1 **my-pipeline-ci** は **Repository** CRD の namespace です。

- 任意のパイプライン実行を対話的に行うには、以下の例を使用します。

```
$ tkn pac logs -n <my-pipeline-ci> 1
```

1 **my-pipeline-ci** は **Repository** CRD の namespace です。最後のパイプライン実行以外のパイプライン実行を表示する必要がある場合は、**tkn pac logs** コマンドを使用して、リポジトリーにアタッチされた **PipelineRun** を選択できます。

GitHub アプリケーションで Pipelines as Code を設定している場合に、Pipelines as Code は GitHub アプリケーションの **Checks** タブで URL を Post します。URL をクリックし、パイプラインの実行をたどることができます。

パイプライン実行の再起動

ブランチへの新しいコミットの送信やプルリクエストの発行など、イベントなしでパイプラインの実行を再開できます。GitHub アプリで、**Checks** タブに移動し、**Re-run** をクリックします。

プルまたはマージ要求をターゲットにする場合は、プル要求内で以下のコメントを使用して、すべてまたは特定のパイプライン実行を再起動します。

- **/retest** コメントは、すべてのパイプラインの実行を再開します。
- **/retest <pipelinerun-name>** コメントは、特定のパイプラインの実行を再開します。
- **/cancel** コメントは、すべてのパイプライン実行をキャンセルします。
- **/cancel <pipelinerun-name>** コメントは、特定のパイプラインの実行をキャンセルします。

コメントの結果は、GitHub アプリケーションの **Checks** タブに表示されます。

4.8.15. Pipelines as Code を使用したパイプライン実行ステータスの監視

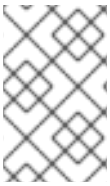
コンテキストおよびサポートされるツールに応じて、パイプライン実行のステータスをさまざまな方法で監視できます。

GitHub アプリケーションのステータス

パイプラインの実行が完了すると、**チェック** タブにステータスが追加され、パイプラインの各タスクにかかった時間に関する情報少しと、**tkn pipelinerun describe** コマンドの出力が表示されます。

ログエラーのスニペット

コードとしてのパイプラインがパイプラインのタスクの1つでエラーを検出すると、最初に失敗したタスクのタスク内訳の最後の 3 行で設定される小さなスニペットが表示されます。



注記

Pipelines as Code は、パイプラインの実行を調べて秘密の値を隠し文字に置き換えることで、シークレットの漏洩を回避します。ただし、Pipelines as Code は、ワークスペースおよび envFrom ソースからのシークレットを非表示にできません。

ログエラースニペットのアノテーション

Pipelines as Code config map で、**error-detection-from-container-logs** パラメーターを **true** に設定すると、Pipelines as Code はコンテナーログからエラーを検出し、エラーが発生したプルリクエストにアノテーションとして追加します。

重要

この機能はテクノロジープレビューです。

現在、Pipelines as Code は、エラーが次の形式の **makefile** または **grep** 出力のように見える単純なケースのみをサポートしています。

```
<filename>:<line>:<column>: <error message>
```

error-detection-simple-regexp フィールドを使用して、エラーの検出に使用される正規表現をカスタマイズできます。正規表現は名前付きグループを使用して、マッチングを柔軟に指定できるようになります。マッチングに必要なグループは filename、line、および error です。デフォルトの正規表現の Pipelines as Code config map を表示できます。

注記

デフォルトでは、コードとしての Pipelines はコンテナーログの最後の 50 行のみをスキャンします。**error-detection-max-number-of-lines** フィールドでこの値を増やすか、**-1** を設定して行数を無制限にすることができます。ただし、このような設定では、ウォッチャーのメモリー使用量が増加する可能性があります。

Webhook のステータス

Webhook の場合、イベントがプルリクエストの場合、ステータスはプルまたはマージリクエストのコメントとして追加されます。

失敗

namespace が **Repository** CRD に一致する場合に、Pipelines as Code は namespace 内の Kubernetes イベントにその失敗ログメッセージを出力します。

Repository CRD に関連付けられたステータス

パイプライン実行の最後の 5 つのステータスメッセージは、**Repository** カスタムリソース内に保存されます。

```
$ oc get repo -n <pipelines-as-code-ci>
```

```
NAME                 URL                                                 NAMESPACE           SUCCEEDED
REASON      STARTTIME   COMPLETIONTIME
pipelines-as-code-ci   https://github.com/openshift-pipelines/pipelines-as-code   pipelines-as-code-ci
True        Succeeded   59m         56m
```

tkn pac describe コマンドを使用すると、リポジトリーおよびそのメタデータに関連付けられた実行のステータスを抽出できます。

通知

Pipelines as Code は通知を管理しません。通知が必要な場合は、パイプラインの **最後** の機能を使用します。

関連情報

- An example task to send Slack messages on success or failure
- An example of a pipeline run with **finally** tasks triggered on push events

4.8.16. Pipelines as Code でのプライベートリポジトリーの使用

Pipelines as Code は、ユーザートークンを使用してターゲット namespace でシークレットを作成または更新することで、プライベートリポジトリーをサポートします。Tekton Hub からの **git-clone** タスクは、ユーザートークンを使用してプライベートリポジトリーのクローンを作成します。

コードとしてのパイプラインは、ターゲット namespace で新しいパイプライン実行を作成するたびに、**pac-gitauth-<REPOSITORY_OWNER>-<REPOSITORY_NAME>-<RANDOM_STRING>** 形式でシークレットを作成または更新します。

パイプライン実行およびパイプライン定義の **basic-auth** ワークスペースでシークレットを参照する必要があり、これは、**git-clone** タスクに渡されます。

```
...
  workspace:
  - name: basic-auth
    secret:
      secretName: "{{ git_auth_secret }}"
...
```

パイプラインでは、**git-clone** タスクの再使用に **basic-auth** ワークスペースを参照できます。

```
...
workspaces:
  - name basic-auth
params:
    - name: repo_url
    - name: revision
...
tasks:
  workspaces:
    - name: basic-auth
      workspace: basic-auth
  ...
```

```
  tasks:
  - name: git-clone-from-catalog
      taskRef:
        name: git-clone (1)
      params:
        - name: url
          value: $(params.repo_url)
        - name: revision
          value: $(params.revision)
...
```

(1) **git-clone** タスクは **basic-auth** ワークスペースを取得し、これを使用してプライベートリポジトリーのクローンを作成します。

Pipelines as Code config map で必要に応じて、**secret-auto-create** フラグを **false** または **true** の値に設定することで、この設定を変更できます。

関連情報

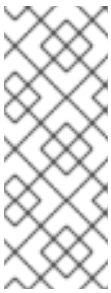
- An example of the **git-clone** task used for cloning private repositories

4.8.17. Pipelines as Code を使用したパイプライン実行のクリーンアップ

ユーザー namespace には多数のパイプラインの実行があります。**max-keep-runs** アノテーションを設定することで、イベントに一致するパイプライン実行を限られた数だけ保持するように Pipelines as Code を設定できます。以下に例を示します。

```
...
  pipelinesascode.tekton.dev/max-keep-runs: "<max_number>" (1)
...
```

(1) Pipelines as Code は、正常な実行の終了直後にクリーンアップを開始し、アノテーションを使用して設定されたパイプライン実行の最大数のみを保持します。



注記

- コードとしてのパイプラインは、実行中のパイプラインのクリーニングをスキップしますが、ステータスが不明のパイプラインの実行をクリーンアップします。
- Pipelines as Code は、失敗したプルリクエストのクリーニングをスキップします。

4.8.18. Pipelines as Code での受信 Webhook の使用

受信 Webhook URL と共有シークレットを使用して、リポジトリーでパイプラインの実行を開始できます。

受信 Webhook を使用するには、**Repository** CRD の **spec** セクション内に以下を指定します。

- Pipelines as Code が一致する受信 Webhook URL。

- Git プロバイダーおよびユーザートークン。現時点で、Pipelines as Code は **github**、**gitlab**、および **bitbucket-cloud** をサポートします。

注記

GitHub アプリケーションのコンテキストで受信 Webhook URL を使用する場合は、トークンを指定する必要があります。

- 受信 Webhook URL のターゲットブランチおよびシークレット。

例: 受信 Webhook のあるリポジトリー CRD

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: repo
  namespace: ns
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    type: github
    secret:
      name: "owner-token"
  incoming:
    - targets:
      - main
      secret:
        name: repo-incoming-secret
      type: webhook-url
```

例: 受信 Webhook のリポジトリーンのシークレット

```
apiVersion: v1
kind: Secret
metadata:
  name: repo-incoming-secret
  namespace: ns
type: Opaque
stringData:
  secret: <very-secure-shared-secret>
```

Git リポジトリーの **.tekton** ディレクトリーにあるパイプライン実行をトリガーするには、以下のコマンドを使用します。

```
$ curl -X POST 'https://control.pac.url/incoming?secret=very-secure-shared-
secret&repository=repo&branch=main&pipelinerun=target_pipelinerun'
```

Pipelines as Code は受信 URL を照合し、それを **push** イベントとして扱います。ただし、Pipelines as Code は、このコマンドによってトリガーされたパイプライン実行のステータスを報告しません。

レポートまたは通知を取得するには、**finally** タスクを使用してこれをパイプラインに直接追加します。または、**tkn pac** CLI ツールを使用して **Repository** CRD を検査できます。

4.8.19. Pipelines as Code 設定のカスタマイズ

クラスター管理者は **pipelines-as-code** namespace の **pipelines-as-code** 設定マップを使用して以下のパラメーターを設定し、Pipelines as Code をカスタマイズすることができます。

表4.8 Pipelines as Code 設定のカスタマイズ

パラメーター	説明	デフォルト
application-name	アプリケーションの名前。たとえば、GitHub Checks ラベルに表示される名前です。	**"Pipelines as Code CI"**
max-keep-days	実行されたパイプライン実行が **pipelines-as-code namespace** に保持される日数。 この configmap の設定は、ユーザーのパイプライン実行のクリーンアップには影響しません。これは、ユーザーの GitHub リポジトリーのパイプライン実行定義のアノテーションによって制御されます。	
secret-auto-create	GitHub アプリケーションで生成されたトークンを使用してシークレットを自動的に作成するかどうかを示します。このシークレットは、プライベートリポジトリーで使用できます。	**enabled**
remote-tasks	有効にすると、パイプライン実行アノテーションからのリモートタスクを許可します。	**enabled**
hub-url	Tekton Hub API のベース URL。	**https://hub.tekton.dev/**
hub-catalog-name	Tekton Hub のカタログ名。	**tekton**
tekton-dashboard-url	Tekton Hub ダッシュボードの URL。Pipelines as Code は、この URL を使用して、Tekton Hub ダッシュボードに **PipelineRun** URL を生成します。	NA
bitbucket-cloud-check-source-ip	パブリック Bitbucket の IP 範囲をクエリーしてサービス要求を保護するかどうかを示します。パラメーターのデフォルト値を変更すると、セキュリティーの問題が発生する可能性があります。	**enabled**

パラメーター	説明	デフォルト
bitbucket-cloud-additional-source-ip	コンマで区切られた追加の IP 範囲またはネットワークのセットを提供するかどうかを示します。	NA
max-keep-run-upper-limit	パイプライン実行の **max-keep-run** 値の上限。	NA
default-max-keep-runs	パイプライン実行の **max-keep-run** 値のデフォルトの制限。定義されている場合、値は **max-keep-run** アノテーションを持たないすべてのパイプライン実行に適用されます。	NA
auto-configure-new-github-repo	新しい GitHub リポジトリーを自動的に設定します。Pipelines as Code は namespace を設定し、リポジトリーのカスタムリソースを作成します。このパラメーターは、GitHub アプリケーションでのみサポートされています。	**disabled**
auto-configure-repo-namespace-template	**auto-configure-new-github-repo** が有効になっている場合は、新しいリポジトリーの namespace を自動的に生成するようにテンプレートを設定します。	**{repo_name}-pipelines**
error-log-snippet	失敗したタスク (パイプラインにエラーがある) のログスニペットの表示を有効または無効にします。パイプラインからのデータ漏えいの場合は、このパラメーターを無効にすることができます。	**enabled**

4.8.20. Pipelines as Code のコマンドリファレンス

tkn pac CLI ツールは、以下の機能を提供します。

- ブートストラップ Pipelines as Code のインストールおよび設定。
- 新規 Pipelines as Code リポジトリーの作成。
- すべての Pipeline as Code リポジトリーをリスト表示。
- Pipeline as Code リポジトリーおよび関連付けられた実行の記述。
- 使用を開始するための単純なパイプライン実行の生成。

- Pipelines as Code によって実行されたかのようにパイプラインの実行を解決。

ヒント

アプリケーションのソースコードが含まれる Git リポジトリーに変更を加える必要がないように、テストおよび実験用に機能に対応するコマンドを使用することができます。

4.8.20.1. 基本的な構文

```
$ tkn pac [command or options] [arguments]
```

4.8.20.2. グローバルオプション

```
$ tkn pac --help
```

4.8.20.3. ユーティリティーコマンド

4.8.20.3.1. bootstrap

表4.9 ブートストラップ Pipelines as Code のインストールおよび設定

コマンド	説明
tkn pac bootstrap	GitHub および GitHub Enterprise などのサービスプロバイダーをホストする Git リポジトリーの Pipelines as Code をインストールおよび設定します。
tkn pac bootstrap --nightly	Pipelines as Code のナイトリービルドをインストールします。
tkn pac bootstrap --route-url <public_url_to_ingress_spec>	OpenShift ルートの URL をオーバーライドします。 デフォルトでは、**tkn pac bootstrap** は OpenShift ルートを検出します。これは、Pipelines as Code コントローラーサービスに自動的に関連付けられます。 OpenShift Container Platform クラスターがない場合、受信エンドポイントをポイントするパブリック URL の入力を要求します。
tkn pac bootstrap github-app	**pipelines-as-code** namespace に GitHub アプリケーションとシークレットを作成します。

4.8.20.3.2. repository

表4.10 Pipelines as Code リポジトリーの管理

コマンド	説明
tkn pac repo create	パイプライン実行テンプレートに基づいて、新規 Pipelines as Code リポジトリーおよび namespace を作成します。
tkn pac repo list	すべての v リポジトリーとしてリスト表示し、関連する実行の最後のステータスを表示します。
tkn pac repo describe	Pipelines as Code リポジトリーおよび関連する実行を記述します。

4.8.20.3.3. generate

表4.11 Pipelines as Code を使用したパイプライン実行の生成

コマンド	説明
tkn pac generate	単純なパイプライン実行を生成します。 ソースコードが含まれるディレクトリーから実行すると、現在の Git 情報を自動的に検出します。 さらに、基本的な言語検出機能を使用して、言語に応じてさらにタスクを追加します。 たとえば、リポジトリーのルートで **setup.py** ファイルを検出すると、pylint タスク が自動的に生成されたパイプライン実行に追加されます。

4.8.20.3.4. resolve

表4.12 Pipelines as Code を使用したパイプライン実行の解決および実行

コマンド	説明
tkn pac resolve	サービスで Pipelines as Code により所有されているかのようにパイプライン実行を実行します。
tkn pac resolve -f .tekton/pull-request.yaml | oc apply -f -	**.tekton/pull-request.yaml** のテンプレートを使用するライブのパイプライン実行のステータスを表示します。 ローカルマシンで実行中の Kubernetes インストールと組み合わせて、新しいコミットを生成せずにパイプライン実行を確認できます。 ソースコードリポジトリーからコマンドを実行すると、現在の Git 情報を検出し、現在のリビジョンやブランチなどのパラメーターを自動的に解決しようとします。

コマンド	説明
tkn pac resolve -f .tekton/pr.yaml -p revision=main -p repo_name= <repository_name>	Git リポジトリーからのデフォルトのパラメーター値をオーバーライドして、パイプライン実行を実行します。 **-f** オプションはディレクトリーパスを受け入れ、そのディレクトリー内のすべての **.yaml** または **.yml** ファイルに **tkn pac resolve** コマンドを適用することもできます。1つのコマンドで **-f** フラグを複数回使用することもできます。 **-p** オプションを使用してパラメーター値を指定することで、Git リポジトリーから収集したデフォルト情報をオーバーライドできます。たとえば、Git ブランチをリビジョンおよび異なるリポジトリー名として使用できます。

4.8.21. 関連情報

- An example of the **.tekton/** directory in the Pipelines as Code repository
- OpenShift Pipelines のインストール
- tkn のインストール
- Red Hat OpenShift Pipelines リリースノート

4.9. WEB コンソールでの RED HAT OPENSHIFT PIPELINES の使用

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

4.9.1. Developer パースペクティブで Red Hat OpenShift Pipelines を使用する

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

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

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

前提条件

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

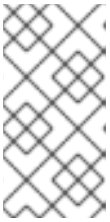
4.9.2. Pipeline Builder を使用した Pipeline の構築

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

- **Pipeline ビルダー** または **YAML ビュー** のいずれかを使用してパイプラインを設定します。
- 既存のタスクおよびクラスタータスクを使用して、パイプラインフローを構築します。OpenShift Pipelines Operator をインストールする際に、再利用可能なパイプラインクラスタータスクをクラスターに追加します。
- パイプライン実行に必要なリソースタイプを指定し、必要な場合は追加のパラメーターをパイプラインに追加します。
- パイプラインの各タスクのこれらのパイプラインリソースを入力および出力リソースとして参照します。
- 必要な場合は、タスクのパイプラインに追加されるパラメーターを参照します。タスクのパラメーターは、Task の仕様に基づいて事前に設定されます。
- Operator によってインストールされた、再利用可能なスニペットおよびサンプルを使用して、詳細なパイプラインを作成します。

手順

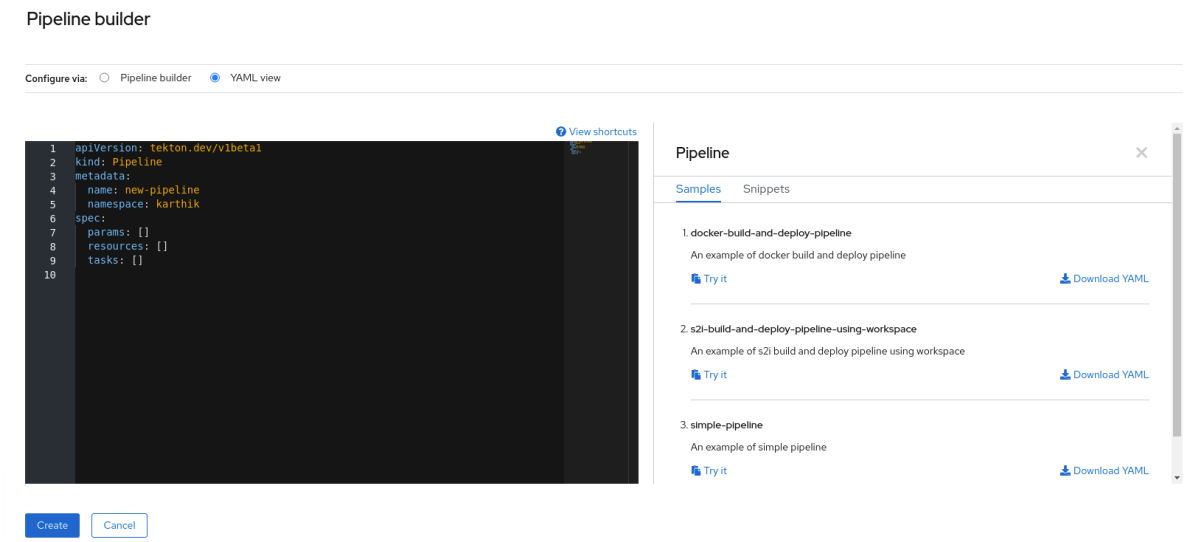
1. **Developer** パースペクティブの **+Add** ビューで、**Pipeline** タイルをクリックし、**Pipeline Builder** ページを表示します。
2. **Pipeline ビルダー** ビューまたは **YAML ビュー** のいずれかを使用して、パイプラインを設定します。



注記

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

図4.1 YAML ビュー



3. **Pipeline builder** を使用してパイプラインを設定します。
   a. **Name** フィールドにパイプラインの一意の名前を入力します。
   b. **Tasks** セクションで、以下を実行します。
      i. **Add task** をクリックします。
      ii. クイック検索フィールドを使用してタスクを検索し、表示されたリストから必要なタスクを選択します。
      iii. **Add** または **Install and add** をクリックします。この例では、**s2i-nodejs** タスクを使用します。

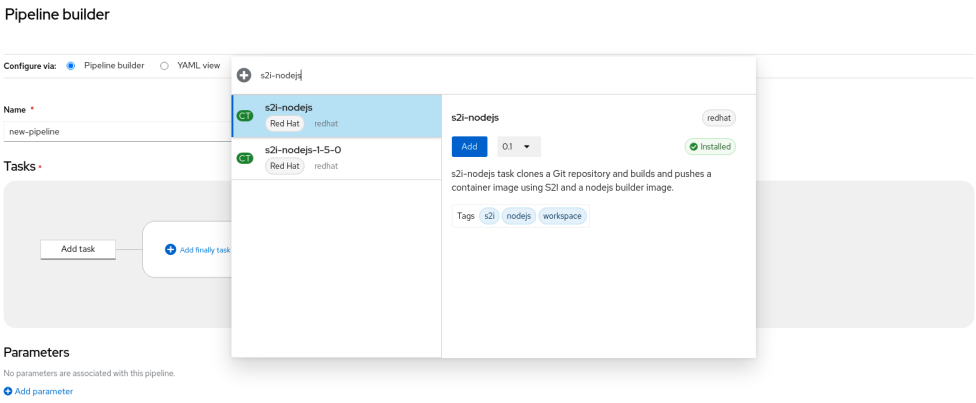
注記

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

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

- Add または Install and add をクリックします。

図4.2 Pipeline Builder

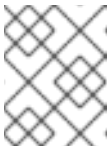


- 最終タスクを追加するには、以下を実行します。
  - Add finally task → Add task の順にクリックします。
  - クイック検索フィールドを使用してタスクを検索し、表示されたリストから必要なタスクを選択します。
  - Add または Install and add をクリックします。

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

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

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



注記

リソースが見つからない場合には、タスクの横に赤のアイコンが表示されます。

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

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

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

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

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

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

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

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

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

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

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

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

Tekton Hub 統合はデフォルトで有効になっており、クラスターでサポートされている Tekton Hub からのタスクを確認できます。管理者は Tekton Hub 統合をオプトアウトでき、Tekton Hub タスクは表示されなくなります。生成されたパイプラインに Webhook URL が存在するかどうかを確認することもできます。**+Add**フローを使用して作成されたパイプラインにデフォルトの Webhook が追加され、Topology ビューで選択したリソースのサイドパネルに URL が表示されます。

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

4.9.4. Developer パースペクティブを使用したパイプラインの使用

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

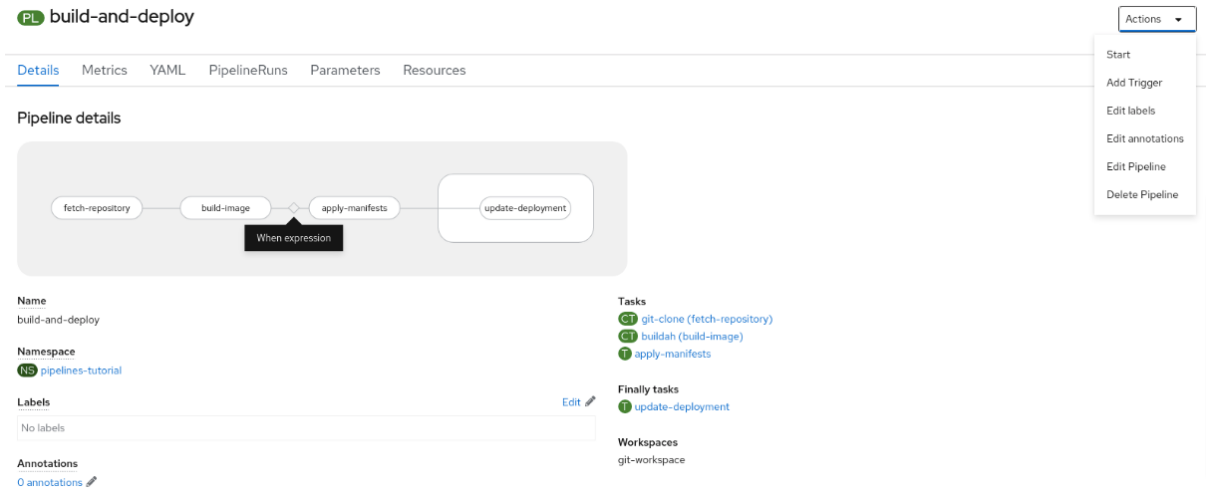
- パイプラインが作成された namespace
- 最後のパイプライン実行
- パイプライン実行のタスクのステータス
- パイプライン実行のステータス
- 最後のパイプライン実行の作成時間

手順

1. **Developer** パースペクティブの **Pipelines** ビューで、**Project** ドロップダウンリストからプロジェクトを選択し、そのプロジェクトのパイプラインを表示します。

2. 必要なパイプラインをクリックし、**Pipeline Details** ページを表示します。
デフォルトでは、**Details** タブには、すべてのシリアルタスク、並列タスク、**finally** タスク、およびパイプラインの式がすべて視覚的に表示されます。タスクと **finally** タスクは、ページの右下にリスト表示されます。リスト表示されている **Tasks** および **Finally** タスクをクリックして、タスクの詳細を表示します。

図4.3 Pipeline の詳細



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

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

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

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

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

     注記

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

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

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

注記

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

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

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

4.9.5. Pipelines ビューからパイプラインを開始する

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

手順

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

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

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

注記

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

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

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

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

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

i. シークレットの一意の **シークレット名**。

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

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

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

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

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

注記

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

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

iv. シークレットを追加するためにチェックマークを選択します。

パイプラインのリソースの数に基づいて、複数のシークレットを追加できます。

4. **Start** をクリックしてパイプラインを開始します。

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

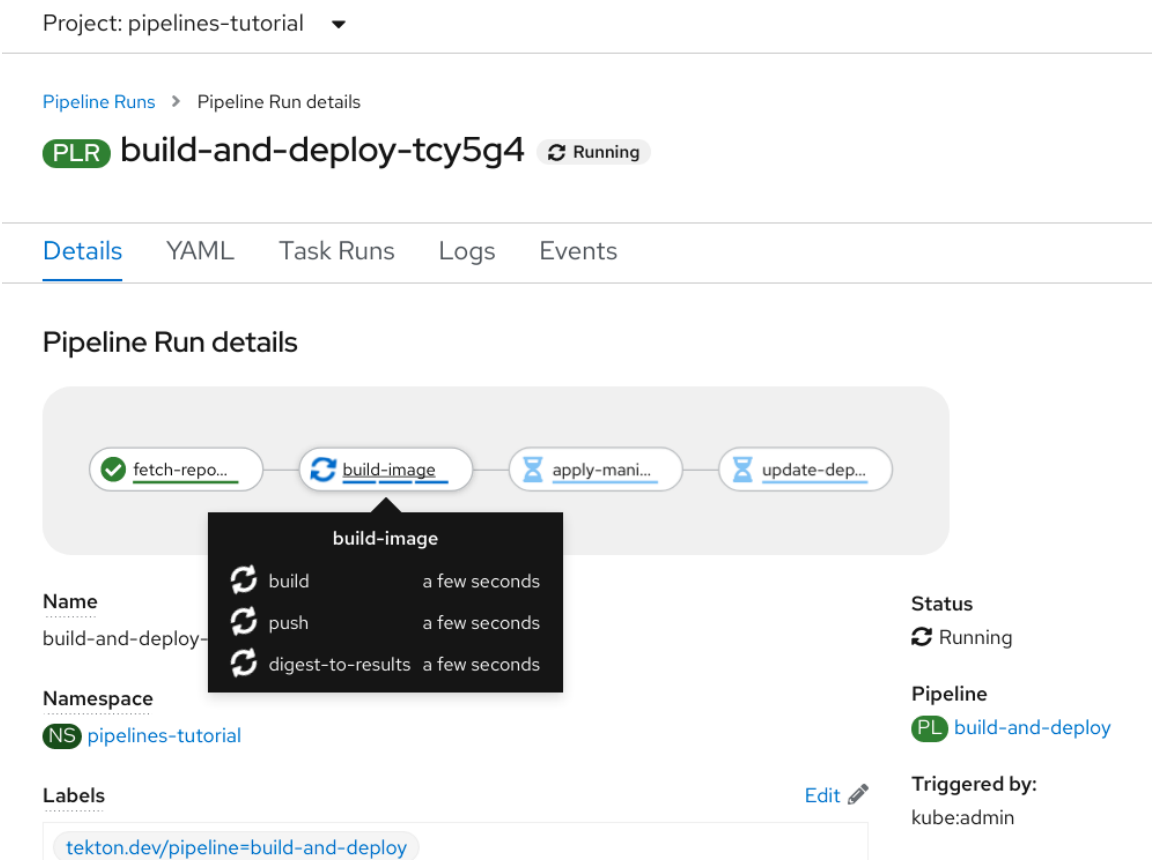
- 各ステップの実行にかかった時間を表示するには、タスクにカーソルを合わせます。

- タスクをクリックし、タスクの各ステップのログを表示します。

- **Logs** タブをクリックして、タスクの実行シーケンスに関連するログを表示します。該当するボタンを使用して、ペインをデプロイメントし、ログを個別に、または一括してダウンロードすることもできます。

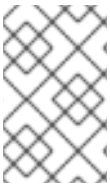
- **Events** タブをクリックして、パイプライン実行で生成されるイベントのストリームを表示します。
  Task Runs、**Logs**、および **Events** タブを使用すると、失敗したパイプラインの実行またはタスクの実行のデバッグに役立ちます。

図4.4 パイプライン実行の詳細



4.9.6. Topology ビューからパイプラインを開始する

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



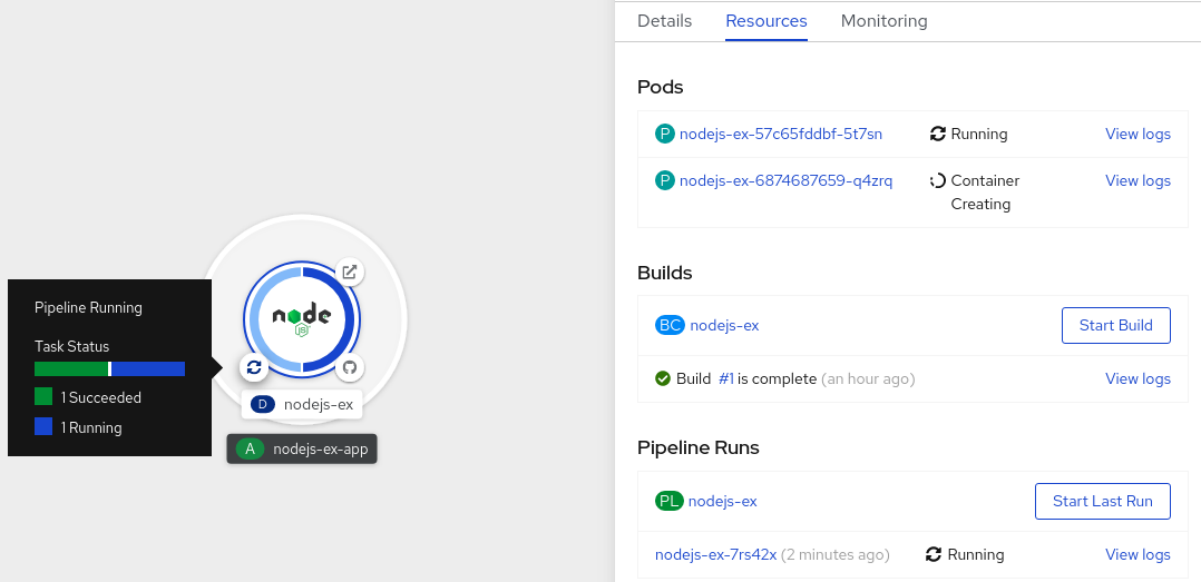
注記

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

手順

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

図4.5 Topology ビューのパイプライン



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

4.9.7. Topology ビューからのパイプラインとの対話

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

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

4.9.8. Pipeline の編集

Web コンソールの **Developer** パースペクティブを使用して、クラスターの 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. **パイプラインビルダー** ページでは、次のタスクを実行できます。
   - 追加のタスク、パラメーター、またはリソースをパイプラインに追加します。

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

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

4.9.9. Pipeline の削除

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

手順

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

4.9.9.1. 関連情報

- パイプラインでの Tekton Hub の使用

4.9.10. Administrator パースペクティブでパイプラインテンプレートを作成する

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

前提条件

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

手順

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

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
# ...
  namespace: openshift (1)
  labels:
```

```
    pipeline.openshift.io/runtime: <runtime> 2
    pipeline.openshift.io/type: <pipeline-type> 3
# ...
```

1. テンプレートは **openshift** namespace に作成する必要があります。
2. テンプレートには **pipeline.openshift.io/runtime** ラベルが含まれている必要があります。このラベルで許可されるランタイム値は、**nodejs**、**golang**、**dotnet**、**java**、**php**、**ruby**、**perl**、**python**、**nginx**、および **httpd** です。
3. テンプレートには、**pipeline.openshift.io/type:** ラベルが含まれている必要があります。このラベルで許可されるタイプ値は、**openshift**、**knative**、および **kubernetes** です。

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

4.10. TEKTONCONFIG カスタムリソース設定のカスタマイズ

Red Hat OpenShift Pipelines では、**TektonConfig** カスタムリソース (CR) を使用して以下の設定をカスタマイズできます。

- Red Hat OpenShift Pipelines コントロールプレーンの設定
- デフォルトサービスアカウントの変更
- サービスモニターの無効化
- クラスタータスクとパイプラインテンプレートの無効化
- Tekton Hub 統合の無効化
- RBAC リソースの自動作成の無効化
- タスク実行とパイプライン実行のプルーニング

4.10.1. 前提条件

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

4.10.2. Red Hat OpenShift Pipelines コントロールプレーンの設定

TektonConfig カスタムリソース (CR) の設定フィールドを編集して、Pipelines コントロールプレーンをカスタマイズできます。Red Hat OpenShift Pipelines Operator は設定フィールドにデフォルト値を自動的に追加し、Pipelines コントロールプレーンを使用可能な状態にします。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Administration → CustomResourceDefinitions** に移動します。
2. **Search by name** ボックスを使用して、**tektonconfigs.operator.tekton.dev** カスタムリソース定義 (CRD) を検索します。**TektonConfig** をクリックし、CRD の詳細ページを表示します。

3. **Instances** タブをクリックします。
4. **config** インスタンスをクリックして、**TektonConfig** CR の詳細を表示します。
5. **YAML** タブをクリックします。
6. 要件に応じて **TektonConfig** YAML ファイルを編集します。

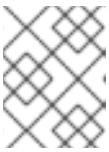
デフォルト値が適用された TektonConfig CR の例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    running-in-environment-with-injected-sidecars: true
    metrics.taskrun.duration-type: histogram
    metrics.pipelinerun.duration-type: histogram
    await-sidecar-readiness: true
    params:
      - name: enableMetrics
        value: 'true'
    default-service-account: pipeline
    require-git-ssh-secret-known-hosts: false
    enable-tekton-oci-bundles: false
    metrics.taskrun.level: task
    metrics.pipelinerun.level: pipeline
    embedded-status: both
    enable-api-fields: stable
    enable-provenance-in-status: false
    enable-custom-tasks: true
    disable-creds-init: false
    disable-affinity-assistant: true
```

4.10.2.1. デフォルト値が適用された変更可能フィールド

次のリストには、デフォルト値が適用された **TektonConfig** CR の変更可能フィールドがすべて含まれています。

- **running-in-environment-with-injected-sidecars** (デフォルト: **true**): Istio などの注入済みサイドカーを使用しないクラスターでパイプラインを実行する場合は、このフィールドを **false** に設定します。**false** に設定すると、パイプラインがタスク実行を開始するまでにかかる時間が短縮されます。



注記

注入されたサイドカーを使用するクラスターの場合、このフィールドを **false** に設定すると、予期しない動作が発生する可能性があります。

- **await-sidecar-readiness** (デフォルト: **true**): **TaskRun** サイドカーコンテナーの実行を待たずに Pipelines が動作を開始するようにするには、このフィールドを **false** に設定します。これにより、**downwardAPI** ボリュームタイプをサポートしない環境でのタスク実行が可能になります。

- **default-service-account** (デフォルト: **Pipeline**): 特に指定されていない場合、このフィールドには **TaskRun** および **PipelineRun** リソースに使用するデフォルトのサービスアカウント名が設定されます。

- **require-git-ssh-secret-known-hosts** (デフォルト: **false**): このフィールドを **true** に設定するには、Git SSH シークレットに **known_hosts** フィールドが含まれている必要があります。

  - Git SSH シークレットの設定について、詳しくは **関連情報** セクションの **Git の SSH 認証の設定** を参照してください。

- **Enable-tekton-oci-bundles** (デフォルト: **false**): このフィールドを **true** に設定すると、Tekton OCI バンドルという名前の実験的アルファ機能の使用が可能になります。

- **embedded-status** (デフォルト: **Both**): このフィールドには、次の 3 つの値を使用できます。

  - **full**: **PipelineRun** ステータスに **Run** ステータスと **TaskRun** ステータスを完全に埋め込めます。

  - **minimal**: **ChildReferences** フィールドに、ステータスが PipelineRun` の実行とタスク実行の情報 (名前、種類、API バージョンなど) を追加します。

  - **both**: **full** と **minimal** の両方の値が適用されます。

注記

embedded-status フィールドは非推奨となり、将来のリリースで削除される予定です。さらに、パイプラインにデフォルトで埋め込まれるステータスは、**minimal** に変更されます。

- **Enable-api-fields** (デフォルト: **stable**): このフィールドを設定すると、どの機能が有効になるかが決まります。使用できる値はは **stable**、**beta**、または **alpha** です。

注記

Red Hat OpenShift Pipelines で **alpha** 値はサポートされていません。

- **Enable-provenance-in-status** (デフォルト: **false**): このフィールドを **true** に設定すると、**TaskRun** ステータスおよび **PipelineRun** ステータスの **provenance** フィールドへの入力が可能になります。**provenance** フィールドには、リモートタスクまたはパイプライン定義の取得元などの、タスク実行およびパイプライン実行で使用されるリソースのメタデータが含まれます。

- **Enable-custom-tasks** (デフォルト: **true**): このフィールドを **false** に設定すると、パイプラインでのカスタムタスクの使用が無効になります。

- **disable-creds-init** (デフォルト: **false**): Pipelines が接続されたサービスアカウントをスキャンしてステップに認証情報を挿入しないようにするには、このフィールドを **true** に設定します。

- **disable-affinity-assistant** (デフォルト: **true**): 永続ボリューム要求ワークスペースを共有する各 **TaskRun** リソースに対してアフィニティーアシスタントを有効にするには、このフィールドを **false** に設定します。

メトリクスオプション

TektonConfig CR の次のメトリクスフィールドのデフォルト値を変更できます。

- **metrics.taskrun.duration-type** と **metrics.pipelinerun.duration-type** (デフォルト: **histogram**): これらのフィールドを設定すると、タスクまたはパイプライン実行の期間のタイプが決まります。使用できる値は、**gauge** または **histogram** です。

- **metrics.taskrun.level** (デフォルト: **task**): このフィールドにより、タスク実行メトリクスのレベルが決まります。使用できる値は、**taskrun**、**task**、または **namespace** です。

- **metrics.pipelinerun.level** (デフォルト: **Pipeline**): このフィールドにより、パイプライン実行メトリクスのレベルが決まります。使用できる値は、**pipelinerun**、**pipeline**、または **namespace** です。

4.10.2.2. 任意の設定フィールド

次のフィールドにはデフォルト値がなく、設定した場合にのみ考慮されます。デフォルトでは、Operator はこれらのフィールドを **TektonConfig** カスタムリソース (CR) に追加も設定もしません。

- **default-timeout-minutes**: **TaskRun** および **PipelineRun** リソースの作成時に指定していない場合、このフィールドがデフォルトのタイムアウトを設定します。タスク実行またはパイプライン実行にかかる時間が設定された分数より長いと、タスク実行またはパイプライン実行はタイムアウトになり、キャンセルされます。たとえば、**default-timeout-minutes: 60** はデフォルトを 60 分に設定します。

- **default-managed-by-label-value**: このフィールドには、**app.kubernetes.io/managed-by** ラベルに指定されたデフォルト値が含まれます。このデフォルト値は、何も指定されていない場合にすべての **TaskRun** Pod に適用されます。たとえば、**default-managed-by-label-value: tekton-pipelines** です。

- **default-pod-template**: このフィールドは、指定されていない場合にデフォルトの **TaskRun** および **PipelineRun** Pod テンプレートを設定します。

- **default-cloud-events-sink**: このフィールドは、何も指定されていない場合に、**TaskRun** および **PipelineRun** リソースに使用されるデフォルトの **CloudEvents** シンクを設定します。

- **default-task-run-workspace-binding**: このフィールドには、**Task** リソースが宣言するワークスペースのデフォルトワークスペース設定が含まれますが、**TaskRun** リソースは明示的に宣言されません。

- **default-affinity-assistant-pod-template**: このフィールドは、何も指定されていない場合にアフィニティーアシスタント Pod が使用するデフォルトの **PipelineRun** Pod テンプレートを設定します。

- **default-max-matrix-combinations-count**: このフィールドには、何も指定されていない場合の、マトリクスから生成される組み合わせの最大数のデフォルト値が含まれます。

4.10.3. Pipelines のデフォルトサービスアカウントの変更

Pipeline のデフォルトサービスアカウントは、**.spec.pipeline** および **.spec.trigger** 仕様の **default-service-account** フィールドを編集して変更できます。デフォルトのサービスアカウントの名前は **pipeline** です。

例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
```

```
spec:
  pipeline:
    default-service-account: pipeline
  trigger:
    default-service-account: pipeline
    enable-api-fields: stable
```

4.10.4. サービスモニターの無効化

Pipeline の一部であるサービスモニターを無効にして、Telemetry データを公開できます。サービスモニターを無効にするには、**TektonConfig** カスタムリソース (CR) の **.spec.pipeline** 仕様で **enableMetrics** パラメーターを **false** に設定します。

例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    params:
      - name: enableMetrics
        value: 'false'
```

4.10.5. クラスタータスクとパイプラインテンプレートの無効化

デフォルトでは、**TektonAddon** カスタムリソース (CR) は、クラスター上の Pipeline と併せて **clusterTasks** および **pipelineTemplates** リソースをインストールします。

clusterTasks および **pipelineTemplates** リソースのインストールを無効にするには、**.spec.addon** 仕様でパラメーターの値を **false** に設定します。さらに、**communityClusterTasks** パラメーターも無効にできます。

例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  addon:
    params:
      - name: clusterTasks
        value: 'false'
      - name: pipelineTemplates
        value: 'false'
      - name: communityClusterTasks
        value: 'true'
```

4.10.6. Tekton Hub 統合の無効化

Web コンソールの **Developer** パースペクティブで Tekton Hub の統合を無効にするには、**TektonConfig** カスタムリソース (CR) の **enable-devconsole-integration** パラメーターを **false** に設定します。

Tekton Hub 無効化の例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  hub:
    params:
      - name: enable-devconsole-integration
        value: false
```

4.10.7. RBAC リソースの自動作成の無効化

Red Hat OpenShift Pipelines Operator のデフォルトインストールは、**^(openshift|kube)-*** 正規表現パターンに一致する namespace を除き、クラスター内のすべての namespace について複数のロールベースアクセス制御 (RBAC) リソースを作成します。これらの RBAC リソースの中で、**pipelines-scc-rolebinding** SCC (security context constraint) のロールバインディングリソースは、関連する **pipelines-scc** SCC に **RunAsAny** 権限があるため、セキュリティー上の問題となる可能性があります。

Red Hat OpenShift Pipelines Operator のインストール後にクラスター全体の RBAC リソースの自動作成を無効にするには、クラスター管理者は、クラスターレベルの **TektonConfig** カスタムリソース (CR) で **createRbacResource** パラメーターを **false** に設定します。

TektonConfig CR の例

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
  - name: createRbacResource
    value: "false"
...
```

警告

クラスター管理者または適切な権限を持つユーザーとして、すべての namespace の RBAC リソースの自動作成を無効にすると、デフォルトの **ClusterTask** リソースは機能しません。**ClusterTask** リソースを機能させるには、それぞれの意図された namespace について RBAC リソースを手動で作成する必要があります。

4.10.8. タスク実行とパイプライン実行の自動プルーニング

古い **TaskRun** オブジェクトと **PipelineRun** オブジェクト、およびそれらの実行されたインスタンスは、アクティブな実行に使用できる物理リソースを占有します。リソースの使用を最適化するために、Red Hat OpenShift Pipelines は、クラスター管理者がさまざまな namespace で未使用のオブジェクトとそのインスタンスを自動的にプルーニングするために使用できるアノテーションを提供します。

注記

アノテーションを指定して自動プルーニングを設定すると、namespace 全体に影響します。namespace で個々のタスク実行とパイプライン実行を選択的に自動プルーニングすることはできません。

4.10.8.1. タスク実行とパイプライン実行を自動的にプルーニングするためのアノテーション

namespace でタスク実行とパイプライン実行を自動的にプルーニングするには、namespace で以下のアノテーションを設定できます。

- **operator.tekton.dev/prune.schedule**: このアノテーションの値が **TektonConfig** カスタムリソース定義で指定された値と異なる場合には、その namespace に新規の cron ジョブが作成されます。
- **operator.tekton.dev/prune.skip**: **true**に設定されている場合、それが設定されている namespace はプルーニングされません。
- **operator.tekton.dev/prune.resources**: このアノテーションではリソースのコンマ区切りの一覧を使用できます。パイプライン実行などの単一リソースをプルーニングするには、このアノテーションを **pipelinerun** に設定します。task run や pipeline run などの複数のリソースをプルーニングするには、このアノテーションを **"taskrun, pipelinerun"** に設定します。
- **operator.tekton.dev/prune.keep**: このアノテーションを使用して、プルーニングなしでリソースを保持します。
- **operator.tekton.dev/prune.keep-since**: このアノテーションを使用して、経過時間をもとにリソースを保持します。このアノテーションの値は、リソースの経過時間 (分単位) と等しくなければなりません。たとえば、6 日以上前に作成されたリソースを保持するには、**keep-since** を **7200** に設定します。

  注記

  keep および **keep-since** アノテーションは同時に使用できません。リソースには、どちらか 1 つだけを使用する必要があります。

- **operator.tekton.dev/prune.strategy**: このアノテーションの値を **keep** または **keep-since** のいずれかに設定します。

たとえば、過去 5 日間に作成されたすべてのタスク実行とパイプライン実行を保持し、古いリソースを削除する次のアノテーションについて考えてみます。

自動プルーニングアノテーションの例

```
...
  annotations:
    operator.tekton.dev/prune.resources: "taskrun, pipelinerun"
    operator.tekton.dev/prune.keep-since: 7200
...
```

4.10.9. 関連情報

- Git の SSH 認証の設定
- バージョン管理されていないクラスタータスクおよびバージョン管理されたクラスタータスクの管理
- リソースを回収するためのオブジェクトのプルーニング
- Administrator パースペクティブでパイプラインテンプレートを作成する

4.11. OPENSHIFT パイプラインのリソース消費の削減

マルチテナント環境でクラスターを使用する場合、各プロジェクトおよび Kubernetes オブジェクトの CPU、メモリー、およびストレージリソースの使用を制御する必要があります。これにより、1 つのアプリケーションがリソースを過剰に消費し、他のアプリケーションに影響を与えるのを防ぐことができます。

結果として作成される Pod に設定される最終的なリソース制限を定義するために、Red Hat OpenShift Pipelines は、それらが実行されるプロジェクトのリソースクォータの制限および制限範囲を使用します。

プロジェクトのリソース消費を制限するには、以下を実行できます。

- リソースクォータを設定し、管理 して、リソースの総消費量を制限します。
- 制限範囲を使用し、リソース消費を制限 します。この対象は、Pod、イメージ、イメージストリームおよび永続ボリューム要求 (PVC) などの特定のオブジェクトのリソース消費です。

4.11.1. パイプラインでのリソース消費について

各タスクは、**Task** リソースの **steps** フィールドで定義された、特定の順序で実行される多数の必須ステップで設定されます。各タスクは Pod として実行され、各ステップは同じ Pod 内のコンテナーとして実行されます。

ステップは一度に 1 つずつ実行されます。タスクを実行する Pod は、タスク内の 1 つのコンテナーイメージ (ステップ) を一度に実行するのに十分なリソースのみを要求するため、タスク内のすべてのステップのリソースは保存されません。

steps 仕様の **Resources** フィールドは、リソース消費の制限を指定します。デフォルトで、CPU、メモリー、および一時ストレージのリソース要求は、**BestEffort** (ゼロ) 値またはそのプロジェクトの制限範囲で設定される最小値に設定されます。

ステップのリソース要求および制限の設定例

```
spec:
  steps:
  - name: <step_name>
    resources:
      requests:
        memory: 2Gi
        cpu: 600m
      limits:
        memory: 4Gi
        cpu: 900m
```

LimitRange パラメーターおよびコンテナーリソース要求の最小値がパイプラインおよびタスクが実行されるプロジェクトに指定される場合、Red Hat OpenShift Pipelines はプロジェクトのすべての **LimitRange** 値を確認し、ゼロではなく最小値を使用します。

プロジェクトレベルでの制限範囲パラメーターの設定例

```
apiVersion: v1
kind: LimitRange
metadata:
  name: <limit_container_resource>
spec:
  limits:
  - max:
      cpu: "600m"
      memory: "2Gi"
    min:
      cpu: "200m"
      memory: "100Mi"
    default:
      cpu: "500m"
      memory: "800Mi"
    defaultRequest:
      cpu: "100m"
      memory: "100Mi"
    type: Container
...
```

4.11.2. パイプラインでの追加のリソース消費を軽減する

Pod 内のコンテナーにリソース制限を設定する場合、OpenShift Container Platform はすべてのコンテナーが同時に実行される際に要求されるリソース制限を合計します。

呼び出されるタスクで一度に 1 つのステップを実行するために必要なリソースの最小量を消費するために、Red Hat OpenShift Pipelines は、最も多くのリソースを必要とするステップで指定される CPU、メモリー、および一時ストレージの最大値を要求します。これにより、すべてのステップのリソース要件が満たされます。最大値以外の要求はゼロに設定されます。

ただしこの動作により、リソースの使用率が必要以上に高くなる可能性があります。リソースクォータを使用する場合、これにより Pod がスケジュールできなくなる可能性があります。

たとえば、スクリプトを使用する 2 つのステップを含むタスクと、リソース制限および要求を定義しないタスクについて考えてみましょう。作成される Pod には 2 つの init コンテナー (エントリーポイントコピー用に 1 つとスクリプトの作成用に 1 つ) と 2 つのコンテナー (各ステップに 1 つ) があります。

OpenShift Container Platform はプロジェクトに設定された制限範囲を使用して、必要なリソース要求および制限を計算します。この例では、プロジェクトに以下の制限範囲を設定します。

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
```

```
    min:
      memory: 500Mi
    type: Container
```

このシナリオでは、各 init コンテナーは要求メモリー 1 Gi (制限範囲の上限) を使用し、各コンテナーは 500 Mi の要求メモリーを使用します。そのため、Pod のメモリー要求の合計は 2 Gi になります。

同じ制限範囲が 10 のステップのタスクで使用される場合、最終的なメモリー要求は 5 Gi になります。これは、各ステップで実際に必要とされるサイズ (500 Mi) よりも大きくなります (それぞれのステップは他のステップの後に実行されるためです)。

そのため、リソースによるリソース消費を減らすには、以下を行います。

- スクリプト機能および同じイメージを使用して、複数の異なるステップを 1 つの大きなステップにグループ化し、特定のタスクのステップ数を減らします。これにより、要求される最小リソースを減らすことができます。
- 相互に独立しており、独立して実行できるステップを、単一のタスクではなく、複数のタスクに分散します。これにより、各タスクのステップ数が減り、各タスクの要求が小さくなるため、スケジューラーはリソースが利用可能になるとそれらを実行できます。

4.11.3. 関連情報

- OpenShift Pipeline のコンピュートリソースクォータの設定
- プロジェクトごとのリソースクォータ
- 制限範囲によるリソース消費の制限
- リソース要求および制限

4.12. OPENSHIFT PIPELINE のコンピュートリソースクォータの設定

Red Hat OpenShift Pipelines の **ResourceQuota** オブジェクトは、namespace ごとのリソース消費の合計を制御します。これを使用して、オブジェクトのタイプに基づき、namespace で作成されたオブジェクトの数量を制限できます。さらに、コンピュートリソースクォータを指定して、namespace で消費されるコンピュートリソースの合計量を制限できます。

ただし、namespace 全体のクォータを設定するのではなく、パイプライン実行で作成される Pod が使用するコンピュートリソースの量を制限できます。現時点で、Red Hat OpenShift Pipelines ではパイプラインのコンピュートリソースクォータを直接指定できません。

4.12.1. OpenShift Pipeline でコンピュートリソース消費を制限する別の方法

パイプラインによるコンピュートリソースの使用量をある程度制御するためには、代わりに、以下のアプローチを検討してください。

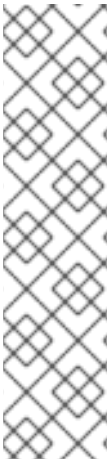
- タスクの各ステップでリソース要求および制限を設定します。

  例: タスクのステップごとのリソース要求および制限設定

```
...
spec:
  steps:
    - name: step-with-limts
```

```
      resources:
        requests:
          memory: 1Gi
          cpu: 500m
        limits:
          memory: 2Gi
          cpu: 800m
...
```

- **LimitRange** オブジェクトの値を指定して、リソース制限を設定します。**LimitRange** の詳細は、制限範囲によるリソース消費の制限 を参照してください。
- パイプラインリソースの消費を減らします。
- プロジェクトごとにリソースクォータ を設定および管理します。
- 理想的には、パイプラインのコンピュートリソースクォータは、パイプライン実行で同時に実行される Pod が消費するコンピュートリソースの合計量と同じである必要があります。ただし、タスクを実行する Pod はユースケースに基づきコンピュートリソースを消費します。たとえば、Maven ビルドタスクには、ビルドするアプリケーションごとに異なるコンピュートリソースが必要となる場合があります。その結果、一般的なパイプラインでタスクのコンピュートリソースクォータを事前に定義できません。コンピュートリソースの使用に関する予測可能性や制御性を高めるには、さまざまなアプリケーション用にカスタマイズされたパイプラインを使用します。



注記

ResourceQuota オブジェクトで設定される namespace で Red Hat OpenShift Pipelines を使用する場合、タスク実行およびパイプライン実行がエラーで失敗する可能性があります (例: **failed quota: <quota name> must specify cpu, memory**)。

このエラーを回避するには、以下のいずれかを実行します。

- (推奨) namespace の制限範囲を指定します。
- すべてのコンテナーの要求および制限を明示的に定義します。

詳細は、問題 および 解決策 を参照してください。

これらの方法で対応できないユースケースには、優先順位クラスのリソースクォータを使用して回避策を実装できます。

4.12.2. 優先順位クラスを使用したパイプラインリソースクォータの指定

PriorityClass オブジェクトは、優先順位クラス名を、相対的な優先順位を示す整数値にマッピングします。値が大きいと、クラスの優先度が高くなります。優先順位クラスの作成後に、仕様に優先順位クラス名を指定する Pod を作成できます。さらに、Pod の優先順位に基づいて、Pod によるシステムリソースの消費を制御できます。

パイプラインにリソースクォータを指定することは、パイプライン実行が作成する Pod のサブセットのリソースクォータを設定することに似ています。以下の手順では、優先順位クラスに基づいてリソースクォータを指定して回避策の例を提供します。

手順

1. パイプラインの優先順位クラスを作成します。

   例: パイプラインの優先順位クラス

   ```
   apiVersion: scheduling.k8s.io/v1
   kind: PriorityClass
   metadata:
     name: pipeline1-pc
   value: 1000000
   description: "Priority class for pipeline1"
   ```

2. パイプラインのリソースクォータを作成します。

   例: パイプラインのリソースクォータ

   ```
   apiVersion: v1
   kind: ResourceQuota
   metadata:
     name: pipeline1-rq
   spec:
     hard:
       cpu: "1000"
       memory: 200Gi
       pods: "10"
     scopeSelector:
       matchExpressions:
       - operator : In
         scopeName: PriorityClass
         values: ["pipeline1-pc"]
   ```

3. パイプラインのリソースクォータの使用量を確認します。

   例: パイプラインにおけるリソースクォータ使用状況の確認

   ```
   $ oc describe quota
   ```

   出力例

   ```
   Name:       pipeline1-rq
   Namespace:  default
   Resource    Used  Hard
   --------    ----  ----
   cpu         0     1k
   memory      0     200Gi
   pods        0     10
   ```

   Pod が実行されていないため、クォータは使用されません。

4. パイプラインおよびタスクを作成します。

   例: パイプラインの YAML

   ```
   apiVersion: tekton.dev/v1alpha1
   kind: Pipeline
   ```

```
metadata:
  name: maven-build
spec:
  workspaces:
  - name: local-maven-repo
  resources:
  - name: app-git
    type: git
  tasks:
  - name: build
    taskRef:
      name: mvn
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["package"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
  - name: int-test
    taskRef:
      name: mvn
    runAfter: ["build"]
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["verify"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
  - name: gen-report
    taskRef:
      name: mvn
    runAfter: ["build"]
    resources:
      inputs:
      - name: source
        resource: app-git
    params:
    - name: GOALS
      value: ["site"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
```

例: パイプラインのタスクの YAML

```
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
```

```
  name: mvn
spec:
  workspaces:
  - name: maven-repo
  inputs:
    params:
    - name: GOALS
      description: The Maven goals to run
      type: array
      default: ["package"]
    resources:
    - name: source
      type: git
  steps:
    - name: mvn
      image: gcr.io/cloud-builders/mvn
      workingDir: /workspace/source
      command: ["/usr/bin/mvn"]
      args:
        - -Dmaven.repo.local=$(workspaces.maven-repo.path)
        - "$(inputs.params.GOALS)"
      priorityClassName: pipeline1-pc
```

注記

パイプラインの全タスクが同じ優先順位クラスに属することを確認します。

5. パイプライン実行を作成して開始します。

例: パイプライン実行の YAML

```
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  generateName: petclinic-run-
spec:
  pipelineRef:
    name: maven-build
  resources:
  - name: app-git
    resourceSpec:
      type: git
      params:
        - name: url
          value: https://github.com/spring-projects/spring-petclinic
```

6. Pod の作成後に、パイプライン実行のリソースクォータの使用状況を確認します。

例: パイプラインにおけるリソースクォータ使用状況の確認

```
$ oc describe quota
```

出力例

```
Name:       pipeline1-rq
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         500m  1k
memory      10Gi  200Gi
pods        1     10
```

この出力は、優先クラスごとにリソースクォータを指定することで、特定の優先クラスに属するすべての同時実行 Pod のリソースクォータをまとめて管理できることを示しています。

4.12.3. 関連情報

- Kubernetes のリソースクォータ
- Kubernetes の制限範囲
- リソース要求および制限

4.13. 特権付きセキュリティーコンテキストでの POD の使用

OpenShift Pipelines 1.3.x 以降のバージョンのデフォルト設定では、パイプライン実行またはタスク実行から Pod が作成される場合、特権付きセキュリティーコンテキストで Pod を実行できません。このような Pod の場合、デフォルトのサービスアカウントは **pipeline** であり、**pipelines** サービスアカウントに関連付けられた SCC (Security Context Constraint) は **pipelines-scc** になります。**pipelines-scc** SCC は **anyuid** SCC と似ていますが、パイプラインの SCC に関する YAML ファイルに定義されるように若干の違いがあります。

pipelines-scc.yaml スニペットの例

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
...
allowedCapabilities:
  - SETFCAP
...
fsGroup:
  type: MustRunAs
...
```

さらに、OpenShift Pipeline の一部として提供される **Buildah** クラスタータスクは、デフォルトのストレージドライバーとして **vfs** を使用します。

4.13.1. 特権付きセキュリティーコンテキストを使用したパイプライン実行 Pod およびタスク実行 Pod の実行

手順

privileged セキュリティーコンテキストで (パイプライン実行またはタスク実行で作成された) Pod を実行するには、以下の変更を行います。

- 関連するユーザーアカウントまたはサービスアカウントを、明示的な SCC を持つように設定します。以下の方法のいずれかを使用して設定を実行できます。

- 以下のコマンドを実行します。

```
$ oc adm policy add-scc-to-user <scc-name> -z <service-account-name>
```

- もしくは、**RoleBinding** および、**Role** または **ClusterRole** の YAML ファイルを変更します。

RoleBinding オブジェクトの例

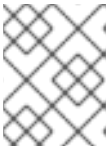
```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-account-name 1
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-clusterrole 2
subjects:
- kind: ServiceAccount
  name: pipeline
  namespace: default
```

1. 適切なサービスアカウント名に置き換えます。
2. 使用するロールバインディングに基づいて適切なクラスターロールに置き換えます。

ClusterRole オブジェクトの例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-clusterrole 1
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use
```

1. 使用するロールバインディングに基づいて適切なクラスターロールに置き換えます。



注記

ベストプラクティスとして、デフォルトの YAML ファイルのコピーを作成し、その複製ファイルに変更を加えます。

- **vfs** ストレージドライバーを使用しない場合、タスク実行またはパイプライン実行に関連付けられたサービスアカウントを特権付き SCC を持つように設定し、セキュリティーコンテキストを **privileged: true** に設定します。

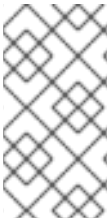
4.13.2. カスタム SCC およびカスタムサービスアカウントを使用したパイプライン実行とタスク実行

デフォルトの **pipelines** サービスアカウントに関連付けられた **pipelines-scc** SCC (Security Context Constraints) を使用する場合、パイプライン実行およびタスク実行 Pod にタイムアウトが生じる可能性があります。これは、デフォルトの **pipelines-scc** SCC で **fsGroup.type** パラメーターが **MustRunAs** に設定されているために発生します。

注記

Pod タイムアウトの詳細は、BZ#1995779 を参照してください。

Pod タイムアウトを回避するには、**fsGroup.type** パラメーターを **RunAsAny** に設定してカスタム SCC を作成し、これをカスタムサービスアカウントに関連付けることができます。



注記

ベストプラクティスとして、パイプライン実行とタスク実行にカスタム SCC およびカスタムサービスアカウントを使用します。このアプローチを使用することで、柔軟性が増し、アップグレード時にデフォルト値が変更されても実行が失敗することはありません。

手順

1. **fsGroup.type** パラメーターを **RunAsAny** に設定してカスタム SCC を定義します。

   例: カスタム SCC

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: my-scc is a close replica of anyuid scc. pipelines-scc has
fsGroup - RunAsAny.
  name: my-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
```

```
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

2. カスタム SCC を作成します。

   例: my-scc SCC の作成

   ```
   $ oc create -f my-scc.yaml
   ```

3. カスタムサービスアカウントを作成します。

   例: fsgroup-runasany サービスアカウントの作成

   ```
   $ oc create serviceaccount fsgroup-runasany
   ```

4. カスタム SCC をカスタムサービスアカウントに関連付けます。

   例: my-scc SCC を fsgroup-runasany サービスアカウントに関連付けます。

   ```
   $ oc adm policy add-scc-to-user my-scc -z fsgroup-runasany
   ```

   特権付きタスクにカスタムサービスアカウントを使用する必要がある場合は、以下のコマンドを実行して **privileged** SCC をカスタムサービスアカウントに関連付けることができます。

   例: fsgroup-runasany サービスアカウントを使用した privileged SCC の関連付け

   ```
   $ oc adm policy add-scc-to-user privileged -z fsgroup-runasany
   ```

5. パイプライン実行およびタスク実行でカスタムサービスアカウントを使用します。

   例: fsgroup-runasany カスタムサービスアカウントを使用した Pipeline 実行 YAML

   ```
   apiVersion: tekton.dev/v1beta1
   kind: PipelineRun
   metadata:
     name: <pipeline-run-name>
   spec:
     pipelineRef:
       name: <pipeline-cluster-task-name>
     serviceAccountName: 'fsgroup-runasany'
   ```

例: fsgroup-runasany カスタムサービスアカウントを使用したタスク実行 YAML

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: <task-run-name>
spec:
  taskRef:
    name: <cluster-task-name>
  serviceAccountName: 'fsgroup-runasany'
```

4.13.3. 関連情報

- SCC の管理についての詳細は、SCC (Security Context Constraints) の管理 を参照してください。

4.14. イベントリスナーによる WEBHOOK のセキュリティー保護

管理者は、イベントリスナーで Webhook をセキュアにできます。namespace の作成後に、**operator.tekton.dev/enable-annotation=enabled** ラベルを namespace に追加して、**Eventlistener** リソースの HTTPS を有効にします。次に、再暗号化した TLS 終端を使用して **Trigger** リソースとセキュアなルートを作成します。

Red Hat OpenShift Pipelines のトリガーは、**Eventlistener** リソースへの非セキュアな HTTP およびセキュアな HTTPS 接続の両方をサポートします。HTTPS は、クラスター内外の接続を保護します。

Red Hat OpenShift Pipelines は、namespace のラベルを監視する **tekton-operator-proxy-webhook** Pod を実行します。ラベルを namespace に追加する場合、Webhook は **service.beta.openshift.io/serving-cert-secret-name=<secret_name>** アノテーションを **EventListener** オブジェクトに設定します。これにより、シークレットおよび必要な証明書が作成されます。

```
service.beta.openshift.io/serving-cert-secret-name=<secret_name>
```

さらに、作成されたシークレットを **Eventlistener** Pod にマウントし、要求を保護できます。

4.14.1. OpenShift ルートを使用したセキュアな接続の提供

再暗号化した TLS 終端を使用してルートを作成するには、以下を実行します。

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

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

セキュアなルートを作成する再暗号化 TLS 終端 YAML の例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: <hostname>
```

```
to:
  kind: Service
  name: frontend (2)
tls:
  termination: reencrypt (3)
  key: [as in edge termination]
  certificate: [as in edge termination]
  caCertificate: [as in edge termination]
  destinationCACertificate: |- (4)
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
```

(1) (2) オブジェクトの名前 (63 文字のみに制限)。

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

(4) これは、再暗号化に必要です。**destinationCACertificate** は CA 証明書を指定してエンドポイントの証明書を検証し、ルーターから宛先 Pod への接続のセキュリティーを保護します。このフィールドは以下のいずれかのシナリオで省略できます。

- サービスは、サービス署名証明書を使用します。
- 管理者はルーターのデフォルト CA 証明書を指定し、サービスにはその CA によって署名された証明書を指定します。

oc create route reencrypt --help コマンドを実行すると、他のオプションを表示できます。

4.14.2. セキュアな HTTPS 接続を使用して EventListener リソースの作成

このセクションでは、pipelines-tutorial の例を使用して、セキュアな HTTPS 接続を使用した EventListener リソースのサンプルの作成について説明します。

手順

1. pipelines-tutorial リポジトリーで利用可能な YAML ファイルから **TriggerBinding** リソースを作成します。

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

2. pipelines-tutorial リポジトリーで利用可能な YAML ファイルから **TriggerTemplate** リソースを作成します。

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

3. **Trigger** リソースを pipelines-tutorial リポジトリーから直接作成します。

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

4. セキュアな HTTPS 接続を使用して **EventListener** リソースの作成します。

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

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

   b. pipelines-tutorial リポジトリーで利用可能な YAML ファイルから **EventListener** リソースを作成します。

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

   c. 再暗号化 TLS 終端でルートを作成します。

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

4.15. GIT シークレットを使用したパイプラインの認証

Git シークレットは、Git リポジトリーと安全に対話するための認証情報で設定されており、認証の自動化に使用されることが多いです。Red Hat OpenShift Pipelines では、Git シークレットを使用して、実行時に Git リポジトリーと対話するパイプライン実行およびタスク実行を認証できます。

パイプライン実行またはタスク実行は、関連付けられたサービスアカウントを介してシークレットにアクセスできます。Pipeline は、Git シークレットの Basic 認証および SSH ベースの認証のアノテーション (キーと値のペア) としての使用をサポートします。

4.15.1. 認証情報の選択

パイプライン実行またはタスク実行には、異なる Git リポジトリーにアクセスするために複数の認証が必要になる場合があります。Pipeline がその認証情報を使用できるドメインで各シークレットにアノテーションを付けます。

Git シークレットの認証情報アノテーションキーは **tekton.dev/git-** で開始する必要があり、その値は Pipeline がその認証情報を使用するホストの URL になります。

以下の例では、Pipeline はユーザー名とパスワードに依存する **basic-auth** シークレットを使用して **github.com** および **gitlab.com** のリポジトリーにアクセスします。

例: Basic 認証用の複数の認証情報

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: github.com
    tekton.dev/git-1: gitlab.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> 1
  password: <password> 2
```

1 リポジトリーのユーザー名

2 リポジトリーのパスワードまたはパーソナルアクセストークン

ssh-auth シークレット (秘密鍵) を使用して Git リポジトリーにアクセスすることもできます。

例: SSH ベースの認証の秘密鍵

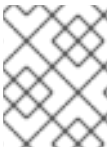
```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 1
```

1 SSH 秘密鍵ファイルの内容。

4.15.2. Git の Basic 認証の設定

パイプラインが、パスワードで保護されたリポジトリーからリソースを取得するには、そのパイプラインの Basic 認証を設定する必要があります。

パイプラインの Basic 認証を設定するには、**secret.yaml**、**serviceaccount.yaml**、および **run.yaml** ファイルを指定されたリポジトリーの Git シークレットからの認証情報で更新します。このプロセスが完了すると、Pipeline はその情報を使用して指定されたパイプラインリソースを取得できます。



注記

GitHub では、プレーンパスワードを使用した認証は非推奨になりました。代わりに、パーソナルアクセストークン を使用します。

手順

1. **secret.yaml** ファイルで、ターゲット Git リポジトリーにアクセスするためのユーザー名とパスワードまたは GitHub パーソナルアクセストークン を指定します。

```
apiVersion: v1
kind: Secret
metadata:
  name: basic-user-pass 1
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
  username: <username> 2
  password: <password> 3
```

1 シークレットの名前。この例では、**basic-user-pass** です。

2 Git リポジトリーのユーザー名。

3 Git リポジトリーのパスワード

2. **serviceaccount.yaml** ファイルで、シークレットを適切なサービスアカウントに関連付けます。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot (1)
secrets:
  - name: basic-user-pass (2)
```

(1) サービスアカウントの名前。この例では、**build-bot** です。

(2) シークレットの名前。この例では、**basic-user-pass** です。

3. **run.yaml** ファイルで、サービスアカウントをタスク実行またはパイプライン実行に関連付けます。

- サービスアカウントをタスク実行に関連付けます。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 (1)
spec:
  serviceAccountName: build-bot (2)
  taskRef:
    name: build-push (3)
```

(1) タスク実行の名前。この例では、**build-push-task-run-2** です。

(2) サービスアカウントの名前。この例では、**build-bot** です。

(3) タスクの名前。この例では、**build-push** です。

- サービスアカウントを **PipelineRun** リソースに関連付けます。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline (1)
  namespace: default
spec:
  serviceAccountName: build-bot (2)
  pipelineRef:
    name: demo-pipeline (3)
```

(1) パイプライン実行の名前。この例では、**demo-pipeline** です。

(2) サービスアカウントの名前。この例では、**build-bot** です。

(3) パイプラインの名前。この例では、**demo-pipeline** です。

4. 変更を適用します。

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.15.3. Git の SSH 認証の設定

パイプラインが SSH キーで設定されたリポジトリーからリソースを取得するには、そのパイプラインの SSH ベースの認証を設定する必要があります。

パイプラインの SSH ベースの認証を設定するには、**secret.yaml**、**serviceaccount.yaml**、および **run.yaml** ファイルを、指定されたリポジトリーの SSH 秘密鍵からの認証情報を使用して更新します。このプロセスが完了すると、Pipeline はその情報を使用して指定されたパイプラインリソースを取得できます。

注記

Basic 認証ではなく SSH ベースの認証を使用することを検討してください。

手順

1. SSH 秘密鍵 を生成するか、既存の秘密鍵をコピーします。これは通常 **~/.ssh/id_rsa** ファイルで入手できます。

2. **secret.yaml** ファイルで、**ssh-privatekey** の値を SSH 秘密鍵ファイルの内容に設定し、**known_hosts** の値を既知のホストファイルの内容に設定します。

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-key (1)
  annotations:
    tekton.dev/git-0: github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: (2)
  known_hosts: (3)
```

(1) SSH 秘密鍵が含まれるシークレットの名前。この例では、**ssh-key** です。

(2) SSH 秘密鍵ファイルの内容。

(3) 既知のホストファイルの内容。

注意

秘密鍵を省略すると、Pipelines は任意のサーバーの公開鍵を受け入れます。

3. オプション: カスタム SSH ポートを指定するには、**annotation** 値の最後に **:<port number>** を追加します。たとえば、**tekton.dev/git-0: github.com:2222** などです。

4. **serviceaccount.yaml** ファイルで、**ssh-key** シークレットを **build-bot** サービスアカウントに関連付けます。

■

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot (1)
secrets:
  - name: ssh-key (2)
```

1. サービスアカウントの名前。この例では、**build-bot** です。
2. SSH 秘密鍵が含まれるシークレットの名前。この例では、**ssh-key** です。

5. **run.yaml** ファイルで、サービスアカウントをタスク実行またはパイプライン実行に関連付けます。
   - サービスアカウントをタスク実行に関連付けます。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 (1)
spec:
  serviceAccountName: build-bot (2)
  taskRef:
    name: build-push (3)
```

1. タスク実行の名前。この例では、**build-push-task-run-2** です。
2. サービスアカウントの名前。この例では、**build-bot** です。
3. タスクの名前。この例では、**build-push** です。

   - サービスアカウントをパイプライン実行に関連付けます。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline (1)
  namespace: default
spec:
  serviceAccountName: build-bot (2)
  pipelineRef:
    name: demo-pipeline (3)
```

1. パイプライン実行の名前。この例では、**demo-pipeline** です。
2. サービスアカウントの名前。この例では、**build-bot** です。
3. パイプラインの名前。この例では、**demo-pipeline** です。

6. 変更を適用します。

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.15.4. git タイプのタスクでの SSH 認証の使用

Git コマンドを呼び出す際には、タスクの手順で直接 SSH 認証を使用できます。SSH 認証は **$HOME** 変数を無視し、**/etc/passwd** ファイルで指定されたユーザーのホームディレクトリーのみを使用します。そのため、タスクの各手順では、**/tekton/home/.ssh** ディレクトリーを、関連付けられたユーザーのホームディレクトリーにシンボリックリンクする必要があります

ただし、**git** タイプのパイプラインリソースまたは Tekton カタログで利用可能な **git-clone** タスクを使用する場合は、明示的なシンボリックリンクは必要ありません。

git タイプのタスクで SSH 認証を使用する例は、 authenticating-git-commands.yaml を参照してください。

4.15.5. root 以外のユーザーとしてのシークレットの使用

以下のような特定のシナリオでは、root 以外のユーザーとしてシークレットを使用する必要がある場合があります。

- コンテナーが実行するために使用するユーザーとグループは、プラットフォームによってランダム化されます。
- タスクの手順では、root 以外のセキュリティーコンテキストを定義します。
- タスクは、root 以外のグローバルセキュリティーコンテキストを指定します。これは、タスクのすべての手順に適用されます。

このようなシナリオでは、root 以外のユーザーとしてタスク実行とパイプライン実行を実行する際の次の側面を考慮してください。

- Git の SSH 認証では、ユーザーが **/etc/passwd** ディレクトリーに有効なホームディレクトリーを設定している必要があります。有効なホームディレクトリーのない UID を指定すると、認証に失敗します。
- SSH 認証は **$HOME** 環境変数を無視します。そのため、Pipelines (**/tekton/home**) で定義される **$HOME** ディレクトリーから、root 以外のユーザーの有効なホームディレクトリーに、適切なシークレットファイルをシンボリックリンクする必要があります。

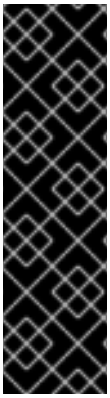
さらに、root 以外のセキュリティーコンテキストで SSH 認証を設定するには、git コマンドを認証する例 を参照してください。

4.15.6. 特定の手順へのシークレットアクセスの制限

デフォルトで、Pipeline のシークレットは **$HOME/tekton/home** ディレクトリーに保存され、タスクのすべての手順で利用できます。

シークレットを特定の手順に制限するには、シークレット定義を使用してボリュームを指定し、特定の手順でボリュームをマウントします。

4.16. OPENSHIFT PIPELINES サプライチェーンセキュリティーでの TEKTON CHAINS の使用



重要

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

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

Tekton Chains は、Kubernetes カスタムリソース定義 (CRD) コントローラーです。これを使用して、Red Hat OpenShift Pipelines を使用して作成されたタスクおよびパイプラインのサプライチェーンセキュリティーを管理できます。

デフォルトでは、Tekton Chains は OpenShift Container Platform クラスター内のすべてのタスク実行を監視します。タスクの実行が完了すると、Tekton Chains はタスク実行のスナップショットを取得します。次に、スナップショットを 1 つ以上の標準ペイロード形式に変換し、最後にすべてのアーティファクトに署名して保存します。

タスクの実行に関する情報を取得するために、Tekton Chains は **Result** オブジェクトと **PipelineResource** オブジェクトを使用します。オブジェクトが使用できない場合、Tekton は OCI イメージの URL と修飾されたダイジェストをチェーンします。

注記

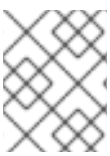
PipelineResource オブジェクトは非推奨であり、将来のリリースで削除される予定です。手動で使用する場合は、**Results** オブジェクトを推奨します。

4.16.1. 主な特長

- 暗号化キータイプと **cosign** などのサービスを使用して、タスク実行、タスク実行結果、および OCI レジストリーイメージに署名できます。
- **in-toto** などの認証形式を使用できます。
- OCI リポジトリーをストレージバックエンドとして使用して、署名と署名されたアーティファクトを安全に保存できます。

4.16.2. Red Hat OpenShift Pipelines Operator を使用した Tekton Chains のインストール

クラスター管理者は、**TektonChain** カスタムリソース (CR) を使用して、Tekton Chains をインストールおよび管理できます。



注記

Tekton Chains は、Red Hat パイプラインのオプションのコンポーネントです。現在、**TektonConfig** を使用してインストールすることはできません。

前提条件

- Red Hat OpenShift Pipelines Operator がクラスターの **openshift-pipelines** namespace にインストールされていることを確認します。

手順

1. OpenShift Container Platform クラスター用の **TektonChain** を作成します。

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonChain
metadata:
  name: chain
spec:
  targetNamespace: openshift-pipelines
```

2. **TektonChain** CR を適用します。

```
$ oc apply -f TektonChain.yaml 1
```

1 **TektonChain** CR のファイル名に置き換えます。

3. インストールのステータスを確認します。

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

4.16.3. Tekton Chains の設定

Tekton Chains は、設定に **openshift-pipelines** namespace で **chains-config** という名前の **ConfigMap** オブジェクトを使用します。

Tekton Chains を設定するには、次の例を使用します。

例: Tekton Chains の設定

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":{"artifacts.oci.storage": "",
"artifacts.taskrun.format":"tekton", "artifacts.taskrun.storage": "tekton"}}' 1
```

1 JSON ペイロードでサポートされているキーと値のペアの組み合わせを使用します。

4.16.3.1. Tekton Chains 設定でサポートされているキー

クラスター管理者は、サポートされているさまざまなキーと値を使用して、タスクの実行、OCI イメージ、およびストレージに関する仕様を設定できます。

4.16.3.1.1. タスク実行でサポートされるキー

表4.13 Chains 設定: タスク実行でサポートされるキー

サポートされているキー	説明	サポート対象の値	デフォルト値
artifacts.taskrun.format	タスク実行ペイロードを格納するためのフォーマット。	**tekton、in-toto**	**tekton**
artifacts.taskrun.storage	タスク実行署名のストレージバックエンド。**“tekton,oci"** のように、複数のバックエンドをコンマ区切りのリストとして指定できます。このアーティファクトを無効にするには、空の文字列 **“"** を指定します。	**tekton、oci**	**tekton**
artifacts.taskrun.signer	タスク実行ペイロードに署名するための署名バックエンド。	**x509**	**x509**

4.16.3.1.2. OCI でサポートされているキー

表4.14 Chains 設定: OCI でサポートされているキー

サポートされているキー	説明	サポート対象の値	デフォルト値
artifacts.oci.format	OCI ペイロードを格納するためのフォーマット。	**simplesigning**	**simplesigning**
artifacts.oci.storage	OCI 署名用のストレージバックエンド。**“oci,tekton"** のように、複数のバックエンドをコンマ区切りのリストとして指定できます。OCI アーティファクトを無効にするには、空の文字列 **“"** を指定します。	**tekton、oci**	**oci**
artifacts.oci.signer	OCI ペイロードに署名するための署名バックエンド。	**x509、cosign**	**x509**

4.16.3.1.3. ストレージ用にサポートされているキー

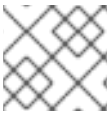
表4.15 Chains 設定: ストレージでサポートされているキー

サポートされているキー	説明	サポート対象の値	デフォルト値
artifacts.oci.repository	OCI 署名を格納するための OCI リポジトリー。	現在、Chains は内部 OpenShift OCI レジストリーのみをサポートしています。Quay などの他の一般的なオプションはサポートされていません。	

4.16.4. Tekton Chains のシークレットに署名する

クラスター管理者は、キーペアを生成し、Tekton Chains を使用して、Kubernetes シークレットを使用してアーティファクトに署名できます。Tekton Chains が機能するには、暗号化されたキーの秘密鍵とパスワードが、**openshift-pipelines** namespace の **signing-secrets** Kubernetes シークレットの一部として存在している必要があります。

現在、Tekton Chains は **x509** および **cosign** 署名スキームをサポートしています。



注記

サポートされている署名スキームの 1 つのみを使用してください。

4.16.4.1. x509 を使用した署名

Tekton Chains で **x509** 署名スキームを使用するには、**ed25519** または **ecdsa** タイプの **x509.pem** 秘密鍵を **signing-secrets** Kubernetes シークレットに保存します。キーが暗号化されていない PKCS8 PEM ファイル (**BEGIN PRIVATE KEY**) として保存されていることを確認します。

4.16.4.2. cosign を使用した署名

Tekton Chains で **cosign** 署名スキームを使用するには:

1. cosign をインストールします。
2. **cosign.key** キーと **cosign.pub** キーのペアを生成します。

   ```
   $ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
   ```

   Cosign はパスワードの入力を求め、Kubernetes シークレットを作成します。
3. 暗号化された **cosign.key** 秘密鍵と **cosign.password** 復号化パスワードを **signing-secrets** Kubernetes シークレットに保存します。秘密鍵が **ENCRYPTED COSIGN PRIVATE KEY** タイプの暗号化された PEM ファイルとして保存されていることを確認します。

4.16.4.3. 署名のトラブルシューティング

署名シークレットがすでに入力されている場合は、次のエラーが発生する可能性があります。

```
Error from server (AlreadyExists): secrets "signing-secrets" already exists
```

エラーを解決するには:

1. シークレットを削除します。

```
$ oc delete secret signing-secrets -n openshift-pipelines
```

2. キーペアを再作成し、好みの署名スキームを使用してシークレットに保存します。

4.16.5. OCI レジストリーへの認証

署名を OCI レジストリーにプッシュする前に、クラスター管理者は、レジストリーで認証するように Tekton Chains を設定する必要があります。Tekton Chains コントローラーは、タスクの実行と同じサービスアカウントを使用します。署名を OCI レジストリーにプッシュするために必要な認証情報を使用してサービスアカウントを設定するには、次の手順を実行します。

手順

1. Kubernetes サービスアカウントの namespace と名前を設定します。

```
$ export NAMESPACE=<namespace> 1
$ export SERVICE_ACCOUNT_NAME=<service_account> 2
```

1 サービスアカウントに関連付けられた namespace。

2 サービスアカウントの名前

2. Kubernetes シークレットを作成します。

```
$ oc create secret registry-credentials \
  --from-file=.dockerconfigjson \ 1
  --type=kubernetes.io/dockerconfigjson \
  -n $NAMESPACE
```

1 Docker 設定ファイルへのパスに置き換えます。デフォルトのパスは **~/.docker/config.json** です。

3. サービスアカウントにシークレットへのアクセス権限を付与します。

```
$ oc patch serviceaccount $SERVICE_ACCOUNT_NAME \
  -p "{\"imagePullSecrets\": [{\"name\": \"registry-credentials\"}]}" -n $NAMESPACE
```

Red Hat OpenShift Pipelines がすべてのタスク実行に割り当てるデフォルトの **pipeline** サービスアカウントにパッチを適用すると、Red Hat OpenShift Pipelines Operator はサービスアカウントをオーバーライドします。ベストプラクティスとして、次の手順を実行できます。

a. ユーザーのタスク実行に割り当てる別のサービスアカウントを作成します。

```
$ oc create serviceaccount <service_account_name>
```

b. タスク実行テンプレートの **serviceaccountname** フィールドの値を設定して、サービスアカウントをタスク実行に関連付けます。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
name: build-push-task-run-2
spec:
serviceAccountName: build-bot 1
taskRef:
  name: build-push
...
```

1 新しく作成したサービスアカウントの名前に置き換えます。

4.16.5.1. 追加認証なしでタスク実行の署名を作成および検証する

追加認証を使用して Tekton Chains でタスク実行の署名を検証するには、次のタスクを実行します。

- 暗号化された x509 キーペアを作成し、Kubernetes シークレットとして保存します。
- Tekton Chains バックエンドストレージを設定します。
- タスク実行を作成して署名し、署名とペイロードをタスク実行自体にアノテーションとして保存します。
- 署名されたタスクの実行から署名とペイロードを取得します。
- タスク実行の署名を確認します。

前提条件

以下がクラスターにインストールされていることを確認します。

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- Cosign

手順

1. 暗号化された x509 鍵ペアを作成し、Kubernetes シークレットとして保存します。

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

プロンプトが表示されたらパスワードを入力します。Cosign は、結果の秘密鍵を **signing-secrets** Kubernetes シークレットの一部として **openshift-pipelines** namespace に保存します。

2. Tekton Chains 設定で、OCI ストレージを無効にし、タスク実行ストレージとフォーマットを **tekton** に設定します。

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":{"artifacts.oci.storage":
"", "artifacts.taskrun.format":"tekton", "artifacts.taskrun.storage": "tekton"}}'
```

3. Tekton Chains コントローラーを再起動して、変更された設定が適用されていることを確認します。

```
$ oc delete po -n openshift-pipelines -l app=tekton-chains-controller
```

4. タスク実行を作成します。

```
$ oc create -f
https://raw.githubusercontent.com/tektoncd/chains/main/examples/taskruns/task-output-image.yaml (1)
taskrun.tekton.dev/build-push-run-output-image-qbjvh created
```

(1) タスクの実行を指す URI またはファイルパスに置き換えます。

5. ステップのステータスを確認し、プロセスが終了するまで待ちます。

```
$ tkn tr describe --last
[...truncated output...]
NAME                                STATUS
· create-dir-builtimage-9467f       Completed
· git-source-sourcerepo-p2sk8       Completed
· build-and-push                    Completed
· echo                              Completed
· image-digest-exporter-xlkn7       Completed
```

6. **base64** でエンコードされたアノテーションとして保存されているオブジェクトから署名とペイロードを取得します。

```
$ export TASKRUN_UID=$(tkn tr describe --last -o  jsonpath='{.metadata.uid}')
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\.tekton\.dev/signature-taskrun-$TASKRUN_UID}" > signature
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\.tekton\.dev/payload-taskrun-$TASKRUN_UID}" | base64 -d > payload
```

7. 署名を確認します。

```
$ cosign verify-blob --key k8s://openshift-pipelines/signing-secrets --signature ./signature ./payload
Verified OK
```

4.16.6. Tekton Chains を使用してイメージと証明書を署名検証する

クラスター管理者は、Tekton Chains を使用して、以下のタスクを実行することで、イメージと証明書を署名および検証できます。

- 暗号化された x509 鍵ペアを作成し、Kubernetes シークレットとして保存します。
- OCI レジストリーの認証を設定して、イメージ、イメージ署名、および署名されたイメージ証明書を保存します。
- Tekton Chains を設定して、証明書を生成し署名します。
- タスク実行で Kaniko を使用してイメージを作成します。

- 署名されたイメージと署名された証明書を検証する。

前提条件

以下がクラスターにインストールされていることを確認します。

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- Cosign
- Rekor
- jq

手順

1. 暗号化された x509 鍵ペアを作成し、Kubernetes シークレットとして保存します。

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

プロンプトが表示されたらパスワードを入力します。Cosign は、結果の秘密鍵を **signing-secrets** Kubernetes シークレットの一部として **openshift-pipelines**namespace に保存し、公開鍵を **cosign.pub** ローカルファイルに書き込みます。

2. イメージレジストリーの認証を設定します。

   a. 署名を OCI レジストリーにプッシュするように Tekton Chains コントローラーを設定するには、タスク実行のサービスアカウントに関連付けられた認証情報を使用します。詳細については、OCI レジストリーへの認証を参照してください。

   b. イメージをビルドしてレジストリーにプッシュする Kaniko タスクの認証を設定するには、必要な認証情報を含む docker **config.json** ファイルの Kubernetes シークレットを作成します。

```
$ oc create secret generic <docker_config_secret_name> \ 1
  --from-file <path_to_config.json> 2
```

1 docker 設定シークレットの名前に置き換えます。

2 docker **config.json** ファイルへのパスに置き換えます。

3. Tekton Chains を設定するには、**chains-config** オブジェクトで **artifacts.taskrun.format**、**artifacts.taskrun.storage**、**transparency.enabled** パラメーターを設定します。

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.format": "in-toto"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.storage": "oci"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"transparency.enabled": "true"}}'
```

4. Kaniko タスクを開始します。

   a. Kaniko タスクをクラスターに適用します。

   ```
   $ oc apply -f examples/kaniko/kaniko.yaml 1
   ```

   1 Kaniko タスクへの URI またはファイルパスに置き換えます。

   b. 適切な環境変数を設定します。

   ```
   $ export REGISTRY=<url_of_registry> 1

   $ export DOCKERCONFIG_SECRET_NAME=
   <name_of_the_secret_in_docker_config_json> 2
   ```

   1 イメージをプッシュするレジストリーの URL に置き換えます。

   2 docker **config.json** ファイルのシークレットの名前に置き換えます。

   c. Kaniko タスクを開始します。

   ```
   $ tkn task start --param IMAGE=$REGISTRY/kaniko-chains --use-param-defaults --
   workspace name=source,emptyDir="" --workspace
   name=dockerconfig,secret=$DOCKERCONFIG_SECRET_NAME kaniko-chains
   ```

   すべての手順が完了するまで、このタスクのログを確認してください。認証が成功すると、最終的なイメージが **$REGISTRY/kaniko-chains** にプッシュされます。

5. Tekton Chains が証明書を生成して署名するまで 1 分ほど待ち、タスク実行時に **chains.tekton.dev/signed=true** アノテーションが利用可能か確認します。

   ```
   $ oc get tr <task_run_name> \ 1
   -o json | jq -r .metadata.annotations

   {
     "chains.tekton.dev/signed": "true",
     ...
   }
   ```

   1 タスク実行の名前に置き換えます。

6. イメージとアテステーションを確認します。

   ```
   $ cosign verify --key cosign.pub $REGISTRY/kaniko-chains

   $ cosign verify-attestation --key cosign.pub $REGISTRY/kaniko-chains
   ```

7. Rekor でイメージの証明書を見つけます。

   a. $ REGISTRY/kaniko-chains イメージのダイジェストを取得します。タスクの実行中に検索するか、イメージをプルしてダイジェストをデプロイメントできます。

b. Rekor を検索して、イメージの **sha256** ダイジェストに一致するすべてのエントリーを見つけます。

```
$ rekor-cli search --sha <image_digest> 1

<uuid_1> 2
<uuid_2> 3
...
```

1 イメージの **sha256** ダイジェストに置き換えます。

2 最初に一致するユニバーサル一意識別子 (UUID)。

3 2 番目に一致する UUID。

検索結果には、一致するエントリーの UUID が表示されます。それらの UUID の 1 つが証明書を保持します。

c. アテステーションを確認してください。

```
$ rekor-cli get --uuid <uuid> --format json | jq -r .Attestation | base64 --decode | jq
```

4.16.7. 関連情報

- OpenShift Pipelines のインストール

4.17. OPENSHIFT LOGGING OPERATOR を使用したパイプラインログの表示

パイプライン実行、タスク実行、およびイベントリスナーによって生成されるログは、それぞれの Pod に保存されます。トラブルシューティングおよび監査に関するログの確認や分析は有用です。

ただし、Pod を無期限に保持すると、リソースを無駄に消費したり、namespace が不必要に分散されたりする可能性があります。

Pod の依存関係を削除して、パイプラインログを表示するには、OpenShift Elasticsearch Operator および OpenShift Logging Operator を使用できます。これらの Operator を使用すると、ログを含む Pod を削除した場合でも、Elasticsearch Kibana スタックを使用してパイプラインログを表示できます。

4.17.1. 前提条件

Kibana ダッシュボードでパイプラインログを表示しようとする前に、以下を確認してください。

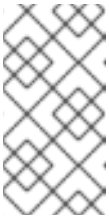
- クラスター管理者がこの手順を実行する。
- パイプライン実行およびタスク実行のログが利用可能である。
- OpenShift Elasticsearch Operator および OpenShift Logging Operator がインストールされている。

4.17.2. Kibana でのパイプラインログの表示

Kibana Web コンソールでパイプラインログを表示するには、以下を実行します。

手順

1. クラスター管理者として OpenShift Container Platform Web コンソールにログインします。
2. メニューバーの右上にある **グリッド** アイコン → **Observability** → **Logging** をクリックします。Kibana Web コンソールが表示されます。
3. インデックスパターンを作成します。
   a. **Kibana** Web コンソールの左側のナビゲーションパネルで **Management** をクリックします。
   b. **Create index pattern** をクリックします。
   c. **ステップ 1/2: Define index pattern**→ **Index pattern** で、* のパターンを入力して **Next Step** をクリックします。
   d. **ステップ 2/2: Configure settings** → **Time filter field name** で、ドロップダウンメニューから **@timestamp** を選択し、**Create index pattern** をクリックします。
4. フィルターを追加します。
   a. **Kibana** Web コンソールの左側のナビゲーションパネルで **Discover** をクリックします。
   b. **Add a filter +**→ **Edit Query DSL** をクリックします。



注記

- 以下のフィルター例の例ごとに、クエリーを編集して **Save** をクリックします。
- フィルターは順次、適用されます。

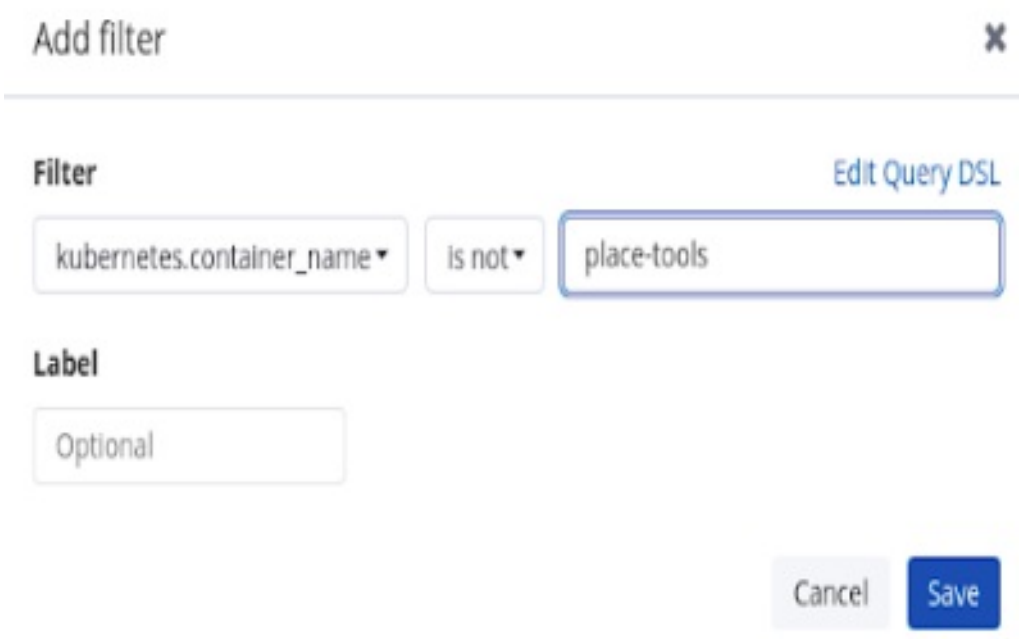
   i. パイプラインに関連するコンテナーをフィルタリングします。

パイプラインコンテナーをフィルタリングするクエリーの例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "app_kubernetes_io/managed-by=tekton-pipelines",
     "type": "phrase"
   }
 }
  }
}
```

   ii. **place-tools** コンテナーではないすべてのコンテナーをフィルタリングします。クエリー DSL を編集する代わりに、グラフィカルドロップダウンメニューを使用する例として、以下の方法を考慮してください。

図4.6 ドロップダウンフィールドを使用したフィルタリングの例



iii. 強調表示できるように **pipelinerun** をラベルでフィルタリングします。

強調表示できるように pipelinerun をラベルでフィルタリングするクエリーの例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "tekton_dev/pipelineRun=",
     "type": "phrase"
   }
 }
  }
}
```

iv. 強調表示できるように **pipeline** をラベルでフィルタリングします。

強調表示できるように pipeline をラベルでフィルタリングするクエリーの例

```
{
  "query": {
 "match": {
   "kubernetes.flat_labels": {
     "query": "tekton_dev/pipeline=",
     "type": "phrase"
   }
 }
  }
}
```

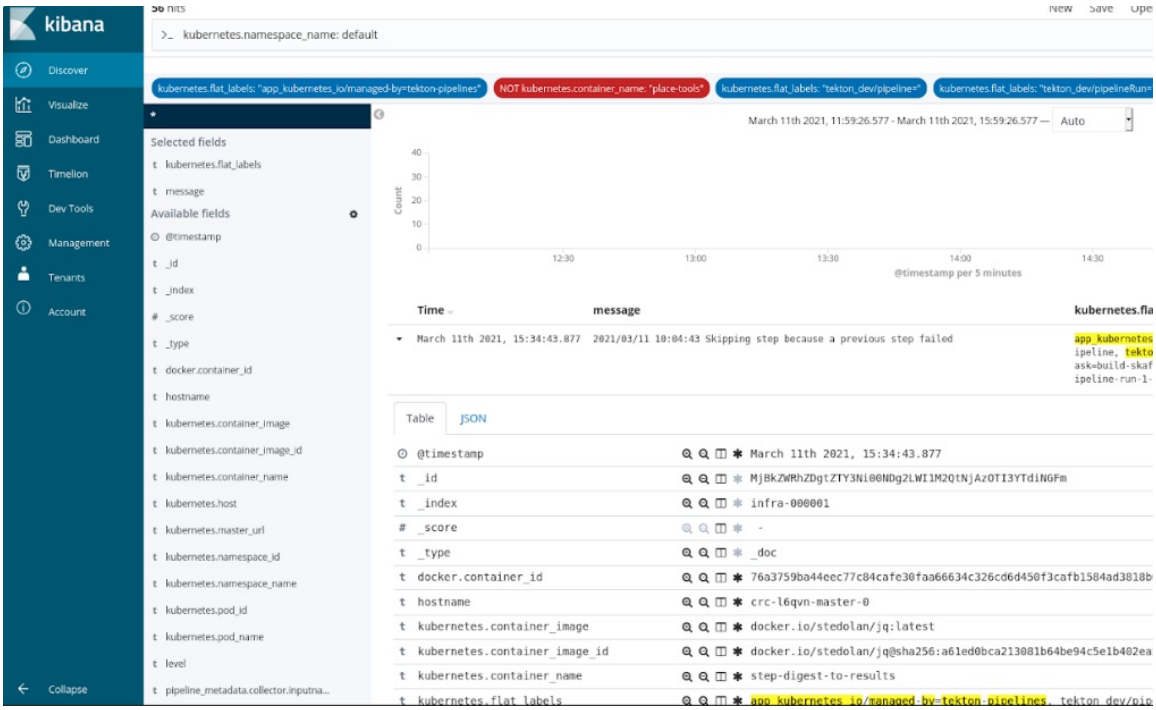
c. **Available fields** リストから以下のフィールドを選択します。

- **kubernetes.flat_labels**

- **message**
  選択したフィールドが **Selected fields** リストに表示されていることを確認します。

d. ログは **message** フィールドの下に表示されます。

図4.7 フィルタリングされたメッセージ



4.17.3. 関連情報

- OpenShift Logging のインストール
- リソースのログの表示
- Kibana を使用したクラスターログの表示

4.18. 非 ROOT ユーザーとして BUILDAH を使用したコンテナーイメージのビルド

コンテナーで root ユーザーとして Pipelines を実行すると、コンテナープロセスとホストが他の悪意のあるリソースにさらされる可能性があります。コンテナー内の特定の root 以外のユーザーとしてワークロードを実行すると、このタイプの露出を減らすことができます。非 root ユーザーとして Buildah を使用してコンテナーイメージのビルドを実行するには、次の手順を実行します。

- カスタムサービスアカウント (SA) とセキュリティーコンテキスト制約 (SCC) を定義します。
- ID が **1000** の **build** ユーザーを使用するように Buildah を設定します。
- カスタム設定マップを使用してタスクの実行を開始するか、パイプラインの実行と統合します。

4.18.1. カスタムサービスアカウントとセキュリティーコンテキストの制約の設定

デフォルトの **pipeline** SA では、namespace の範囲外のユーザー ID を使用できます。デフォルト SA への依存を減らすために、ユーザー ID **1000** の **build** ユーザーに必要なクラスターロールとロールバインディングを使用して、カスタム SA と SCC を定義できます。

重要

現時点で、Buildah がコンテナー内で正常に実行されるために は、**allowPrivilegeEscalation** 設定を有効にする必要があります。この設定により、Buildah は非 root ユーザーとして実行するときに **SETUID** および **SETGID** 機能を活用できます。

手順

- 必要なクラスターロールとロールバインディングを使用して、カスタム SA と SCC を作成します。

例: 使用される ID が 1000 のカスタム SA および SCC

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-sa-userid-1000 1
---
kind: SecurityContextConstraints
metadata:
  annotations:
  name: pipelines-scc-userid-1000 2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true 3
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
- KILL
runAsUser: 4
  type: MustRunAs
  uid: 1000
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
```

```
- projected
- secret
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-userid-1000-clusterrole 5
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - pipelines-scc-userid-1000
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pipelines-scc-userid-1000-rolebinding 6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-userid-1000-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-sa-userid-1000
```

1. カスタム SA を定義します。
2. **runAsUser** フィールドを変更して、制限された権限に基づいて作成されたカスタム SCC を定義します。
3. 現時点で、Buildah がコンテナー内で正常に実行されるためには、**allowPrivilegeEscalation** 設定を有効にする必要があります。この設定により、Buildah は非 root ユーザーとして実行するときに **SETUID** および **SETGID** 機能を活用できます。
4. カスタム SA を介してカスタム SCC にアタッチされた Pod を、ユーザー ID が **1000** として実行されるように制限します。
5. カスタム SCC を使用するクラスターロールを定義します。
6. カスタム SCC を使用するクラスターロールをカスタム SA にバインドします。

4.18.2. build ユーザーを使用するための Buildah の設定

ユーザー ID が **1000** の **build** ユーザーを使用する Buildah タスクを定義できます。

手順

1. **buildah** クラスタータスクのコピーを通常のタスクとして作成します。

```
$ oc get clustertask buildah -o yaml | yq '. |= (del .metadata |= with_entries(select(.key ==
"name" )))' | yq '.kind="Task"' | yq '.metadata.name="buildah-as-user"' | oc create -f -
```

2. コピーした **buildah** タスクを編集します。

```
$ oc edit task buildah-as-user
```

例: build ユーザーで変更された Buildah タスク

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: buildah-as-user
spec:
  description: >-
    Buildah task builds source into a container image and
    then pushes it to a container registry.
    Buildah Task builds source into a container image using Project Atomic's
    Buildah build tool.It uses Buildah's support for building from Dockerfiles,
    using its buildah bud command.This command executes the directives in the
    Dockerfile to assemble a container image, then pushes that image to a
    container registry.
  params:
  - name: IMAGE
    description: Reference of the image buildah will produce.
  - name: BUILDER_IMAGE
    description: The location of the buildah builder image.
    default:
registry.redhat.io/rhel8/buildah@sha256:99cae35f40c7ec050fed3765b2b27e0b8bbea2aa2da7
c16408e2ca13c60ff8ee
  - name: STORAGE_DRIVER
    description: Set buildah storage driver
    default: vfs
  - name: DOCKERFILE
    description: Path to the Dockerfile to build.
    default: ./Dockerfile
  - name: CONTEXT
    description: Path to the directory to use as context.
    default: .
  - name: TLSVERIFY
    description: Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)
    default: "true"
  - name: FORMAT
    description: The format of the built container, oci or docker
    default: "oci"
  - name: BUILD_EXTRA_ARGS
    description: Extra parameters passed for the build command when building images.
    default: ""
  - description: Extra parameters passed for the push command when pushing images.
    name: PUSH_EXTRA_ARGS
    type: string
    default: ""
  - description: Skip pushing the built image
    name: SKIP_PUSH
    type: string
```

```
  default: "false"
results:
- description: Digest of the image just built.
  name: IMAGE_DIGEST
  type: string
workspaces:
- name: source
steps:
- name: build
  securityContext:
    runAsUser: 1000 (1)
  image: $(params.BUILDER_IMAGE)
  workingDir: $(workspaces.source.path)
  script: |
    echo "Running as USER ID `id`" (2)
    buildah --storage-driver=$(params.STORAGE_DRIVER) bud \
      $(params.BUILD_EXTRA_ARGS) --format=$(params.FORMAT) \
      --tls-verify=$(params.TLSVERIFY) --no-cache \
      -f $(params.DOCKERFILE) -t $(params.IMAGE) $(params.CONTEXT)
    [[ "$(params.SKIP_PUSH)" == "true" ]] && echo "Push skipped" && exit 0
    buildah --storage-driver=$(params.STORAGE_DRIVER) push \
      $(params.PUSH_EXTRA_ARGS) --tls-verify=$(params.TLSVERIFY) \
      --digestfile $(workspaces.source.path)/image-digest $(params.IMAGE) \
      docker://$(params.IMAGE)
    cat $(workspaces.source.path)/image-digest | tee /tekton/results/IMAGE_DIGEST
  volumeMounts:
  - name: varlibcontainers
    mountPath: /home/build/.local/share/containers (3)
volumes:
- name: varlibcontainers
  emptyDir: {}
```

(1) Buildah イメージの **build** ユーザーに対応し、明示的に ID が **1000** のユーザーとして指定してコンテナーを実行します。

(2) ユーザー ID を表示して、プロセスがユーザー ID **1000** として実行されていることを確認します。

(3) 必要に応じて、ボリュームマウントのパスを変更できます。

4.18.3. カスタムの config map を使用したタスク実行またはパイプライン実行の開始

カスタム Buildah クラスタータスクを定義したら、ユーザー ID が **1000** の **build** ユーザーとしてイメージをビルドする **TaskRun** オブジェクトを作成できます。さらに、**TaskRun** オブジェクトを **PipelineRun** オブジェクトの一部として統合できます。

手順

1. カスタム **ConfigMap** および **Dockerfile** オブジェクトを使用して **TaskRun** オブジェクトを作成します。

   例: Buildah をユーザー ID 1000 として実行するタスク実行

```
apiVersion: v1
```

```
data:
  Dockerfile: |
    ARG BASE_IMG=registry.access.redhat.com/ubi8/ubi
    FROM $BASE_IMG AS buildah-runner
    RUN dnf -y update && \
        dnf -y install git && \
        dnf clean all
    CMD git
kind: ConfigMap
metadata:
  name: dockerfile (1)
---
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: buildah-as-user-1000
spec:
  serviceAccountName: pipelines-sa-userid-1000 (2)
  params:
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  taskRef:
    kind: Task
    name: buildah-as-user
  workspaces:
  - configMap:
      name: dockerfile (3)
    name: source
```

1. Dockerfile を使用してソースを取得するなどの事前タスクはなく、タスクの実行に焦点が置かれているため、config map を使用します。
2. 作成したサービスアカウントの名前。
3. **buildah-as-user** タスクのソースワークスペースとして config map をマウントします。

2. (オプション) パイプラインと対応するパイプライン実行を作成します。

例: パイプラインと対応するパイプラインの実行

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-buildah-as-user-1000
spec:
  params:
  - name: IMAGE
  - name: URL
  workspaces:
  - name: shared-workspace
  - name: sslcertdir
    optional: true
  tasks:
  - name: fetch-repository (1)
```

```
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.URL)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
  - name: buildah
    taskRef:
      name: buildah-as-user 2
    runAfter:
    - fetch-repository
    workspaces:
    - name: source
      workspace: shared-workspace
    - name: sslcertdir
      workspace: sslcertdir
    params:
    - name: IMAGE
      value: $(params.IMAGE)
---
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: pipelinerun-buildah-as-user-1000
spec:
  taskRunSpecs:
    - pipelineTaskName: buildah
      taskServiceAccountName: pipelines-sa-userid-1000 3
  params:
  - name: URL
    value: https://github.com/openshift/pipelines-vote-api
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  pipelineRef:
    name: pipeline-buildah-as-user-1000
  workspaces:
  - name: shared-workspace 4
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 100Mi
```

1 **git-clone** クラスタータスクを使用して、Dockerfile を含むソースを取得し、変更された Buildah タスクを使用してそれをビルドします。

2 変更された Buildah タスクを参照してください。

3 Buildah タスク用に作成したサービスアカウントを使用します。

4 コントローラーによって自動的に作成される永続ボリューム要求 (PVC) を使用して、**git-clone** タスクと変更された Buildah タスクの間でデータを共有します。

3. タスクの実行またはパイプラインの実行を開始します。

4.18.4. 非特権ビルドの制限

非特権ビルドのプロセスは、ほとんどの **Dockerfile** オブジェクトで機能します。ただし、ビルドが失敗する原因となる既知の制限がいくつかあります。

- **--mount=type=cache** オプションの使用は、必要となる権限の問題が原因で失敗する場合があります。詳細は、この記事 を参照してください。
- **--mount=type=secret** オプションの使用は失敗します。リソースのマウントには、カスタム SCC によって提供されない追加の機能が必要になるためです。

関連情報

- SSC (Security Context Constraints) の管理

第5章 GITOPS

5.1. RED HAT OPENSHIFT GITOPS リリースノート

Red Hat OpenShift GitOps は、クラウドネイティブアプリケーションの継続的デプロイメントを実装するための宣言的な方法です。Red Hat OpenShift GitOps は、異なる環境 (開発、ステージ、実稼働環境など) の異なるクラスターにアプリケーションをデプロイする場合に、アプリケーションの一貫性を確保します。Red Hat OpenShift GitOps は、以下のタスクを自動化する上で役立ちます。

- クラスターに設定、モニタリングおよびストレージについての同様の状態があることの確認。
- クラスターを既知の状態からのリカバリーまたは再作成。
- 複数の OpenShift Container Platform クラスターに対する設定変更を適用するか、これを元に戻す。
- テンプレート化された設定の複数の異なる環境への関連付け。
- ステージから実稼働環境へと、クラスター全体でのアプリケーションのプロモート。

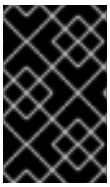
Red Hat OpenShift GitOps の概要については、OpenShift GitOps について を参照してください。

5.1.1. 互換性およびサポート表

現在、今回のリリースに含まれる機能には テクノロジープレビュー のものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

以下の表では、機能は以下のステータスでマークされています。

- **TP: テクノロジープレビュー機能**
- **GA: 一般公開機能**
- **NA: 該当なし**



重要

OpenShift Container Platform 4.13 では、**stable** チャネルが削除されました。OpenShift Container Platform 4.13 にアップグレードする前に、すでに **stable** チャネルを使用している場合は、適切なチャネルを選択してそれに切り替えます。

OpenShift GitOps	コンポーネントのバージョン							OpenShift のバージョン
バージョン	kam	Helm	Kustomize	Argo CD	ApplicationSet	Dex	RH SSO	
1.8.0	0.0.47 TP	3.10.0 GA	4.5.7 GA	2.6.3 GA	NA	2.35.1 GA	7.5.1 GA	4.10 - 4.13

OpenShift GitOps	コンポーネントのバージョン							OpenShift のバージョン
1.7.0	0.0.46 TP	3.10.0 GA	4.5.7 GA	2.5.4 GA	NA	2.35.1 GA	7.5.1 GA	4.10 - 4.12
1.6.0	0.0.46 TP	3.8.1 GA	4.4.1 GA	2.4.5 GA	一般提供され、ArgoCD コンポーネントに含まれています	2.30.3 GA	7.5.1 GA	4.8-4.11
1.5.0	0.0.42 TP	3.8.0 GA	4.4.1 GA	2.3.3 GA	0.4.1 TP	2.30.3 GA	7.5.1 GA	4.8-4.11
1.4.0	0.0.41 TP	3.7.1 GA	4.2.0 GA	2.2.2 GA	0.2.0 TP	2.30.0 GA	7.4.0 GA	4.7-4.10
1.3.0	0.0.40 TP	3.6.0 GA	4.2.0 GA	2.1.2 GA	0.2.0 TP	2.28.0 GA	7.4.0 GA	4.7 - 4.9、4.6 (限定的な GA サポート)
1.2.0	0.0.38 TP	3.5.0 GA	3.9.4 GA	2.0.5 GA	0.1.0 TP	NA	7.4.0 GA	4.8
1.1.0	0.0.32 TP	3.5.0 GA	3.9.4 GA	2.0.0 GA	NA	NA	NA	4.7

- **kam** は、Red Hat OpenShift GitOps Application Manager コマンドラインインターフェイス (CLI) です。
- RH SSO は、Red Hat SSO の略です。

5.1.1.1. テクノロジープレビューの機能

次の表に記載されている機能は、現在テクノロジープレビュー (TP) です。これらの実験的機能は、実稼働環境での使用を目的としていません。

表5.1 テクノロジープレビュートラッカー

機能	Red Hat OpenShift GitOps バージョンの TP	Red Hat OpenShift GitOps バージョンの GA
ApplicationSet プログレッシブロールアウト戦略	1.8.0	NA
アプリケーションの複数のソース	1.8.0	NA
コントロールプレーン以外の namespace の Argo CD アプリケーション	1.7.0	NA
Argo CD 通知コントローラー	1.6.0	NA
OpenShift Container Platform Web コンソールの **Developer** パースペクティブの Red Hat OpenShift GitOps **Environments** ページ	1.1.0	NA

5.1.2. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

5.1.3. Red Hat OpenShift GitOps 1.8.4 のリリースノート

Red Hat OpenShift GitOps 1.8.4 が OpenShift Container Platform 4.10、4.11、4.12、および 4.13 で利用できるようになりました。

5.1.3.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- 今回の更新により、同梱の Argo CD がバージョン 2.6.13 に更新されました。

5.1.3.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新前は、namespace とアプリケーションが増えると、Argo CD が応答しなくなることがありました。リソースを獲得するために機能が競合するため、デッドロックが発生しました。この更新では、デッドロックを削除することで問題を修正します。現在は、namespace やアプリケーションが増えても、クラッシュや応答不能は発生しません。GITOPS-3192

- この更新前は、アプリケーションを再同期するときに Argo CD アプリケーションコントローラーリソースが突然動作を停止することがありました。今回の更新では、クラスターキャッシュのデッドロックを防ぐロジックを追加することで問題を修正しました。これで、アプリケーションは正常に再同期されるはずです。GITOPS-3052

- この更新前は、**argocd-ssh-known-hosts-cm** config map 内の既知のホストの RSA キーが一致しませんでした。今回の更新では、RSA キーをアップストリームプロジェクトと一致させることで問題を修正しました。現在は、デフォルトのデプロイメントでデフォルトの RSA キーを使

用できます。GITOPS-3144

- この更新の前は、Red Hat OpenShift GitOps Operator をデプロイするときに古い Redis イメージバージョンが使用されていたため、脆弱性が発生していました。この更新では、Redis を **registry.redhat.io/rhel-8/redis-6** イメージの最新バージョンにアップグレードすることで、Redis の脆弱性を修正します。GITOPS-3069

- この更新が行われる前は、ユーザーは Operator によってデプロイメントされた Argo CD を介して Microsoft Team Foundation Server (TFS) タイプの Git リポジトリーに接続できませんでした。この更新では、Operator の Git バージョンを 2.39.3 に更新することで問題が修正されます。リポジトリー設定中に **Force HTTP basic auth** フラグを設定して、TFS タイプの Git リポジトリーに接続できるようになりました。GITOPS-1315

5.1.3.3. 既知の問題

- 現在、Red Hat OpenShift GitOps 1.8.4 は、OpenShift Container Platform 4.10 および 4.11 の **latest** チャネルでは利用できません。**latest** チャネルは GitOps 1.9.z によって採用されており、これは OpenShift Container Platform 4.12 以降のバージョンでのみリリースされます。
回避策として、**gitops-1.8** チャネルに切り替えて新しい更新を入手します。GITOPS-3158

5.1.4. Red Hat OpenShift GitOps 1.8.3 のリリースノート

Red Hat OpenShift GitOps 1.8.3 が OpenShift Container Platform 4.10、4.11、4.12、および 4.13 で利用できるようになりました。

5.1.4.1. エラータの更新

5.1.4.1.1. RHBA-2023:3206 および RHSA-2023:3229 - Red Hat OpenShift GitOps 1.8.3 セキュリティー更新アドバイザリー

発行日: 2023-05-18

このリリースに含まれるセキュリティー修正のリストは、次のアドバイザリーに記載されています。

- RHBA-2023:3206

- RHSA-2023:3229

Red Hat OpenShift GitOps Operator をインストールしている場合は、次のコマンドを実行して、このリリースのコンテナーイメージを表示します。

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.4.2. 修正された問題

- この更新前は、**Autoscale** が有効になっており、水平 Pod オートスケーラー (HPA) コントローラーがサーバーデプロイメントのレプリカ設定を編集しようとすると、オペレーターがそれを上書きしていました。さらに、autoscaler パラメーターに指定された変更はクラスター上の HPA に正しく伝播されませんでした。今回の更新でこの問題が修正されています。Operator は、**Autoscale** が無効で HPA パラメーターが正しく更新された場合にのみ、レプリカドリフトで調整されるようになりました。GITOPS-2629

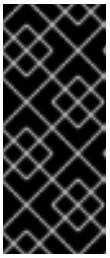
5.1.5. Red Hat OpenShift GitOps 1.8.2 のリリースノート

Red Hat OpenShift GitOps 1.8.2 は、OpenShift Container Platform 4.10、4.11、4.12、4.13 で利用できるようになりました。

5.1.5.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前に、**.spec.dex** パラメーターを使用して Dex を設定し、**LOG IN VIA OPENSHIFT** オプションを使用して Argo CD UI にログインしようとすると、ログインできませんでした。今回の更新でこの問題が修正されています。



重要

ArgoCD CR の **spec.dex** パラメーターは非推奨です。Red Hat OpenShift GitOps v1.9 の将来のリリースでは、ArgoCD CR の **spec.dex** パラメーターを使用した Dex の設定は削除される予定です。代わりに **.spec.sso** パラメーターの使用を検討してください。.spec.sso を使用した Dex の有効化または無効化を参照してください。GITOPS-2761

- 今回の更新前は、OpenShift Container Platform 4.10 クラスターに Red Hat OpenShift GitOps v1.8.0 を新規インストールすると、クラスターおよび **kam** CLI Pod の起動に失敗していました。今回の更新で問題が修正され、すべての Pod が期待どおりに動作するようになりました。GITOPS-2762

5.1.6. Red Hat OpenShift GitOps 1.8.1 のリリースノート

Red Hat OpenShift GitOps 1.8.1 が OpenShift Container Platform 4.10、4.11、4.12、および 4.13 で利用できるようになりました。

5.1.6.1. エラータの更新

5.1.6.1.1. RHSA-2023:1452 - Red Hat OpenShift GitOps 1.8.1 セキュリティー更新アドバイザリー

発行: 2023-03-23

このリリースに含まれるセキュリティー修正のリストは RHSA-2023:1452 アドバイザリーに記載されています。

Red Hat OpenShift GitOps Operator をインストールしている場合は、次のコマンドを実行して、このリリースのコンテナーイメージを表示します。

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.7. Red Hat OpenShift GitOps 1.8.0 のリリースノート

Red Hat OpenShift GitOps 1.8.0 が OpenShift Container Platform 4.10、4.11、4.12、および 4.13 で利用できるようになりました。

5.1.7.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- 今回の更新では、ApplicationSet プログレッシブロールアウト戦略機能のサポートを追加でき

ます。この機能を使用すると、ArgoCD ApplicationSet リソースを拡張して、ApplicationSet 仕様またはアプリケーションテンプレートを変更した後に、漸進的なアプリケーションリソース更新のロールアウト戦略を組み込むことができます。この機能を有効にすると、アプリケーションは同時にではなく、宣言された順序で更新されます。GITOPS-956

重要

ApplicationSet プログレッシブロールアウト戦略は、テクノロジープレビュー機能です。

- 今回の更新では、OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Application environments** ページは、Red Hat OpenShift GitOps Application Manager コマンドラインインターフェイス (CLI) の **kam** から切り離されます。環境が OpenShift Container Platform Web コンソールの **Developer** パースペクティブに表示されるように、**kam** CLI を使用して、Application Environment マニフェストを生成する必要はありません。独自のマニフェストを使用できますが、環境は引き続き namespace で表す必要があります。さらに、特定のラベルとアノテーションが必要です。GITOPS-1785

- 今回の更新では、Red Hat OpenShift GitOps Operator および **kam** CLI が OpenShift Container Platform の ARM アーキテクチャーで使用できるようになりました。GITOPS-1688

重要

spec.sso.provider: keycloak は ARM ではまだサポートされていません。

- 今回の更新では、**.spec.monitoring.enabled** フラグの値を **true** に設定すると、特定の Argo CD インスタンスのワークロード監視を有効にすることができます。その結果、Operator は各 Argo CD コンポーネントのアラートルールを含む **PrometheusRule** オブジェクトを作成します。これらのアラートルールは、対応するコンポーネントのレプリカ数が一定時間望ましい状態から逸脱した場合にアラートをトリガーします。Operator は、ユーザーが **PrometheusRule** オブジェクトに加えた変更を上書きしません。GITOPS-2459

- 今回の更新では、Argo CD CR を使用して、コマンド引数をリポジトリーサーバーのデプロイに渡すことができます。GITOPS-2445
以下に例を示します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  repo:
    extraRepoCommandArgs:
      - --max.combined.directory.manifests.size
      - 10M
```

5.1.7.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 今回の更新の前は、**ARGOCD_GIT_MODULES_ENABLED** 環境変数を設定できるのは、**openshift-gitops-repo-server** Pod のみであり、**ApplicationSet Controller** Pod では、設定できませんでした。その結果、Git ジェネレーターを使用すると、変数が **ApplicationSet Controller** 環境にないため、子アプリケーションの生成中に Git サブモジュールが複製されま

した。さらに、これらのサブモジュールのクローンを作成するために必要な認証情報が ArgoCD で設定されていない場合、アプリケーションの生成は失敗しました。今回の更新で問題が修正されました。Argo CD CR を使用して、**ArgoCD_GIT_MODULES_ENABLED** などの環境変数を **ApplicationSet Controller** Pod に追加できるようになりました。その後、**ApplicationSet Controller** Pod は、複製されたリポジトリーから子アプリケーションを正常に生成し、その過程でサブモジュールは複製されません。GITOPS-2399
以下に例を示します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  applicationSet:
    env:
     - name: ARGOCD_GIT_MODULES_ENABLED
       value: "true"
```

- 今回の更新の前は、Red Hat OpenShift GitOps Operator v1.7.0 のインストール中に、Dex を認証するために作成されたデフォルトの **argocd-cm.yml** config map ファイルには、base64 でエンコードされたクライアントシークレットが **key:value** ペアの形式で含まれていました。今回の更新では、デフォルトの **argocd-cm.yml** config map ファイルにクライアントシークレットを保存しないことで、この問題が修正されています。代わりに、クライアントシークレットは **argocd-secret** オブジェクト内にあり、設定マップ内でシークレット名として参照できます。GITOPS-2570

5.1.7.3. 既知の問題

- **kam** CLI を使用せずに、マニフェストを使用して、アプリケーションをデプロイし、OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Application environments** ページでアプリケーションを表示すると、カード内の Argo CD アイコンから期待どおりに該当アプリケーションの Argo CD URL がページを読み込まないという問題がありました。GITOPS-2736

5.1.8. Red Hat OpenShift GitOps 1.7.4 のリリースノート

Red Hat OpenShift GitOps 1.7.4 が OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

5.1.8.1. エラータの更新

5.1.8.1.1. RHSA-2023:1454 - Red Hat OpenShift GitOps 1.7.4 セキュリティー更新アドバイザリー

発行: 2023-03-23

このリリースに含まれるセキュリティー修正のリストは RHSA-2023:1454 アドバイザリーに記載されています。

Red Hat OpenShift GitOps Operator をインストールしている場合は、次のコマンドを実行して、このリリースのコンテナーイメージを表示します。

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.9. Red Hat OpenShift GitOps 1.7.3 のリリースノート

Red Hat OpenShift GitOps 1.7.3 は、OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

5.1.9.1. エラータの更新

5.1.9.1.1. RHSA-2023:1454 - Red Hat OpenShift GitOps 1.7.3 セキュリティー更新アドバイザリー

発行: 2023-03-23

このリリースに含まれるセキュリティー修正のリストは RHSA-2023:1454 アドバイザリーに記載されています。

Red Hat OpenShift GitOps Operator をインストールしている場合は、次のコマンドを実行して、このリリースのコンテナーイメージを表示します。

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.10. Red Hat OpenShift GitOps 1.7.1 のリリースノート

Red Hat OpenShift GitOps 1.7.1 は、OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

5.1.10.1. エラータの更新

5.1.10.1.1. RHSA-2023:0467 - Red Hat OpenShift GitOps 1.7.1 セキュリティー更新アドバイザリー

発行日: 2023-01-25

このリリースに含まれるセキュリティー修正のリストは、RHSA-2023:0467 アドバイザリーに記載されています。

Red Hat OpenShift GitOps Operator をインストールしている場合は、次のコマンドを実行して、このリリースのコンテナーイメージを表示します。

```
$ oc describe deployment gitops-operator-controller-manager -n openshift-operators
```

5.1.11. Red Hat OpenShift GitOps 1.7.0 のリリースノート

Red Hat OpenShift GitOps 1.7.0 は、OpenShift Container Platform 4.10、4.11、および 4.12 で利用できるようになりました。

5.1.11.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- 今回の更新により、環境変数を Notifications コントローラーに追加できるようになりました。GITOPS-2313
- 今回の更新により、デフォルトの nodeSelector **"kubernetes.io/os": "linux"** キーと値のペアがすべてのワークロードに追加され、Linux ノードでのみスケジュールが設定されるようになりました。さらに、任意のカスタムノードセレクターがデフォルトに追加され、同じキーを持つ

場合に優先されます。GITOPS-2215

- 今回の更新により、**GitopsService** カスタムリソースを編集することで、Operator ワークロードにカスタムノードセレクターを設定できるようになりました。GITOPS-2164
- 今回の更新により、RBAC ポリシーマッチャーモードを使用して、**glob** (デフォルト) および **regex** のオプションから選択できるようになりました。GITOPS-1975
- 今回の更新では、次の追加のサブキーを使用してリソースの動作をカスタマイズできます。

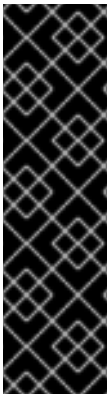
サブキー	キーフォーム	argocd-cm のマップされたフィールド
resourceHealthChecks	resource.customizations.health.<group_kind>	resource.customizations.health
resourceIgnoreDifferences	resource.customizations.ignoreDifferences.<group_kind>	resource.customizations.ignoreDifferences
resourceActions	resource.customizations.actions.<group_kind>	resource.customizations.actions

GITOPS-1561

注記

将来のリリースでは、サブキーではなく resourceCustomization のみを使用してリソースの動作をカスタマイズする古い方法を廃止する可能性があります。

- 今回の更新で、1.7 より前の Red Hat OpenShift GitOps バージョンと OpenShift Container Platform 4.15 以降を使用している場合は、**Developer** パースペクティブで **Environments** ページを使用するには、アップグレードする必要があります。GITOPS-2415
- 今回の更新により、同じクラスター内の任意の namespace で同じコントロールプレーンの Argo CD インスタンスによって管理されるアプリケーションを作成できるようになりました。管理者として以下のアクションを実行し、この更新を有効にします。
  - アプリケーションを管理するクラススコープの Argo CD インスタンスの **.spec.sourceNamespaces** 属性に namespace を追加します。
  - アプリケーションに関連付けられた **AppProject** カスタムリソースの **.spec.sourceNamespaces** 属性に namespace を追加します。GITOPS-2341



重要

コントロールプレーン以外の namespace の Argo CD アプリケーション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 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- 今回の更新により、Argo CD は Server-Side Apply 能をサポートするようになりました。この機能は、ユーザーが以下のタスクを実行するのに役立ちます。
  - 許容されるアノテーションサイズ (262144 バイト) に対して大きすぎる巨大なリソースの管理
  - Argo CD によって管理またはデプロイされていない既存のリソースへのパッチ適用
    この機能は、アプリケーションまたはリソースレベルで設定できます。GITOPS-2340

5.1.11.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前に、Red Hat OpenShift GitOps リリースは、**anyuid** SCC が Dex サービスアカウントに割り当てられたときに **CreateContainerConfigError** エラーで Dex Pod が失敗するという問題の影響を受けていました。この更新プログラムでは、デフォルトのユーザー ID を Dex コンテナーに割り当てることで、この問題を修正しています。GITOPS-2235
- この更新の前は、Red Hat OpenShift GitOps は Dex に加えて OIDC を介して RHSSO (Keycloak) を使用していました。ただし、最近のセキュリティー修正により、有名な認証局のいずれかによって署名されていない証明書で設定されている場合は、RHSSO の証明書を検証できませんでした。この更新で問題が修正されました。カスタム証明書を提供して、通信中に KeyCloak の TLS 証明書を検証できるようになりました。さらに、**rootCA** を Argo CD カスタムリソース **.spec.keycloak.rootCA** フィールドに追加できます。Operator はそのような変更を調整し、**oidc.config in argocd-cm** 設定マップを PEM エンコードされたルート証明書で更新します。GITOPS-2214

Keycloak 設定の Argo CD の例:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  sso:
    keycloak:
      rootCA: '<PEM encoded root certificate>'
    provider: keycloak
.......
.......
```

- この更新の前は、ライブネスプローブが応答しないため、アプリケーションコントローラーが複数回再起動していました。この更新は、アプリケーションコントローラーの **statefulset** アプリケーションで liveness プローブを削除することにより、問題を修正します。GITOPS-2153

5.1.11.3. 既知の問題

- この更新の前に、Operator はリポジトリーサーバーの **mountsatoken** と **ServiceAccount** の設定を調整しませんでした。これは修正されていますが、サービスアカウントを削除してもデフォルトに戻りません。GITOPS-1873
- 回避策: **spec.repo.serviceaccountfield to thedefault** サービスアカウントを手動で設定します。GITOPS-2452

5.1.12. Red Hat OpenShift GitOps 1.6.7 のリリースノート

Red Hat OpenShift GitOps 1.6.7 が OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.12.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新が行われる前は、v0.5.0 以降の Argo CD Operator のすべてのバージョンに情報漏えいの欠陥が存在しました。その結果、権限のないユーザーが API エラーメッセージを検査してアプリケーション名を列挙し、発見されたアプリケーション名を別の攻撃の開始点として使用する可能性があります。たとえば、攻撃者はアプリケーション名に関する知識を利用して、管理者に高い権限を付与するよう説得する可能性があります。この更新により、CVE-2022-41354 エラーが修正されます。GITOPS-2635、CVE-2022-41354

5.1.13. Red Hat OpenShift GitOps 1.6.6 のリリースノート

Red Hat OpenShift GitOps 1.6.6 が OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.13.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新が行われる前は、v0.5.0 以降の Argo CD Operator のすべてのバージョンに情報漏えいの欠陥が存在しました。その結果、権限のないユーザーが API エラーメッセージを検査してアプリケーション名を列挙し、発見されたアプリケーション名を別の攻撃の開始点として使用する可能性があります。たとえば、攻撃者はアプリケーション名に関する知識を利用して、管理者に高い権限を付与するよう説得する可能性があります。この更新により、CVE-2022-41354 エラーが修正されます。GITOPS-2635、CVE-2022-41354

5.1.14. Red Hat OpenShift GitOps 1.6.4 のリリースノート

Red Hat OpenShift GitOps 1.6.4 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.14.1. 修正された問題

- この更新の前は、Argo CD v1.8.2 以降のすべてのバージョンは、不適切な認証バグに対して脆弱でした。その結果、Argo CD はクラスターへのアクセスを目的としていない可能性のあるオーディエンスのトークンを受け入れていました。この問題は修正されています。CVE-2023-22482

5.1.15. Red Hat OpenShift GitOps 1.6.2 のリリースノート

Red Hat OpenShift GitOps 1.6.2 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.15.1. 新機能

- このリリースでは、**openshift-gitops-operator** CSV ファイルから **DISABLE_DEX** 環境変数が削除されています。その結果、Red Hat OpenShift GitOps の新規インストールの実行時にこの環境変数は設定されなくなりました。GITOPS-2360

5.1.15.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、プロジェクトに 5 つを超える Operator がインストールされていると、**InstallPlan** が欠落しているため、サブスクリプションのヘルスチェックは **degraded** とマークされていました。今回の更新でこの問題が修正されています。GITOPS-2018

- この更新の前は、Red Hat OpenShift GitOps Operator は、Argo CD インスタンスが非推奨のフィールドを使用していることを検出すると、非推奨通知の警告をクラスターに送信していました。今回の更新でこの問題が修正され、フィールドを検出したインスタンスごとに警告イベントが 1 つだけ表示されるようになりました。GITOPS-2230

- OpenShift Container Platform 4.12 以降、コンソールのインストールはオプションです。この修正により、Red Hat OpenShift GitOps Operator が更新され、コンソールがインストールされていない場合に Operator でエラーが発生するのを防ぐことができます。GITOPS-2352

5.1.16. Red Hat OpenShift GitOps 1.6.1 のリリースノート

Red Hat OpenShift GitOps 1.6.1 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.16.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、ライブネスプローブが応答しないため、多数のアプリケーションでアプリケーションコントローラーが複数回再起動されていました。この更新は、アプリケーションコントローラーの **StatefulSet** オブジェクトで liveness プローブを削除することにより、問題を修正します。GITOPS-2153

- この更新の前は、証明機関によって署名されていない証明書を使用してセットアップされていると、RHSSO 証明書を検証できませんでした。今回の更新で問題が修正され、通信時に Keycloak の TLS 証明書を検証する際に使用されるカスタム証明書を提供できるようになりました。**rootCA** を Argo CD カスタムリソース **.spec.keycloak.rootCA** フィールドに追加できます。Operator はこの変更を調整し、**argocd-cm ConfigMap** の **oidc.config** フィールドを PEM エンコードされたルート証明書で更新します。GITOPS-2214

注記

.spec.keycloak.rootCA フィールドを更新した後、Argo CD サーバー Pod を再起動します。

以下に例を示します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
     rootCA: |
       ---- BEGIN CERTIFICATE ----
       This is a dummy certificate
       Please place this section with appropriate rootCA
       ---- END CERTIFICATE ----
  server:
    route:
      enabled: true
```

- この更新の前は、Argo CD に管理されていた namespace が終了すると、ロールの作成や他の管理された namespace のその他の設定がブロックされていました。今回の更新でこの問題は修正されています。GITOPS-2277

- この更新の前は、**anyuid** の SCC が Dex **ServiceAccount** リソースに割り当てられている場合、Dex Pod は **CreateContainerConfigError** で開始できませんでした。この更新プログラムでは、デフォルトのユーザー ID を Dex コンテナーに割り当てることで、この問題を修正しています。GITOPS-2235

5.1.17. Red Hat OpenShift GitOps 1.6.0 のリリースノート

Red Hat OpenShift GitOps 1.6.0 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.17.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- 以前は、Argo CD **ApplicationSet** コントローラーはテクノロジープレビュー (TP) 機能でした。この更新により、これは一般提供 (GA) 機能になります。GITOPS-1958

- 今回の更新により、Red Hat OpenShift GitOps の最新リリースが **latest** のバージョンベースのチャネルで利用できるようになりました。これらのアップグレードを取得するには、**Subscription** オブジェクト YAML ファイルの **channel** パラメーターを更新します。値を **stable** から **latest** または **gitops-1.6** などのバージョンベースのチャンネルに変更します。GITOPS-1791

- 今回の更新により、keycloak 設定を制御する **spec.sso** フィールドのパラメーターが **.spec.sso.keycloak** に移動されるようになりました。**.spec.dex** フィールドのパラメーターが **.spec.sso.dex** に追加されました。**.spec.sso.provider** の使用を開始して、Dex を有効または無効にします。**.spec.dex** パラメーターは非推奨であり、キークローク設定の **DISABLE_DEX** および **.spec.sso** フィールドとともに、バージョン 1.9 で削除される予定です。GITOPS-1983

- 今回の更新により、Argo CD カスタムリソースの **.spec.notifications.enabled** パラメーターを使用して有効または無効にできるオプションのワークロードとして、Argo CD 通知コントローラーが利用できるようになりました。Argo CD 通知コントローラーは、テクニカルプレビュー

機能として利用できます。GITOPS-1917

重要

Argo CD Notifications コントローラーはテクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

- 今回の更新により、Tekton パイプライン実行およびタスク実行のリソース除外がデフォルトで追加されました。Argo CD は、デフォルトでこれらのリソースを削除します。これらのリソースの除外は、OpenShift Container Platform から作成された新しい Argo CD インスタンスに追加されます。インスタンスが CLI から作成された場合、リソースは追加されません。GITOPS-1876

- 今回の更新で、Operand の仕様に **resourceTrackingMethod** パラメーターを設定して、Argo CD が使用する追跡方法を選択できるようになりました。GITOPS-1862

- 今回の更新により、Red Hat OpenShift GitOps Argo CD カスタムリソースの **extraConfig** フィールドを使用して、**argocd-cm** configMap にエントリーを追加できるようになりました。指定されたエントリーは、検証なしでライブ **config-cm** configMap に調整されます。GITOPS-1964

- 今回の更新により、OpenShift Container Platform 4.11 では、Red Hat OpenShift GitOps **Developer** パースペクティブの Red Hat OpenShift GitOps **Environments** ページに、アプリケーション環境の成功したデプロイメントの履歴と、各デプロイメントのリビジョンへのリンクが表示されます。GITOPS-1269

- 今回の更新により、Operator によってテンプレートリソースまたはソースとしても使用されている Argo CD を使用してリソースを管理できるようになりました。GITOPS-982

- 今回の更新により、Operator は Kubernetes 1.24 に対して有効にされた Pod Security Admission に対応するために、適切なパーミッションで Argo CD ワークロードを設定するようになりました。GITOPS-2026

- 今回の更新により、Config Management Plugins 2.0 がサポートされるようになりました。Argo CD カスタムリソースを使用して、リポジトリーサーバーのサイドバーコンテナーを指定できます。GITOPS-776

- 今回の更新により、Argo CD コンポーネントと Redis キャッシュ間のすべての通信は、最新の TLS 暗号化を使用して適切に保護されます。GITOPS-720

- Red Hat OpenShift GitOps のこのリリースでは、IBM Z および IBM Power on OpenShift Container Platform 4.10 のサポートが追加されています。現在、制限された環境でのインストールは、IBM Z および IBM Power ではサポートされていません。

5.1.17.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前に、**system:serviceaccount:argocd:gitops-argocd-application-controller** は、

namespace **webapps-dev** の API グループ **monitoring.coreos.com** でリソース prometheusrules を作成できません。今回の更新でこの問題が修正され、Red Hat OpenShift GitOps は **monitoring.coreos.com** API グループからすべてのリソースを管理できるようになりました。GITOPS-1638

- この更新の前に、クラスターのアクセス許可を調整しているときに、シークレットがクラスター設定インスタンスに属している場合、それは削除されていました。今回の更新でこの問題は修正されています。現在は、シークレットの代わりにシークレットの **namespaces** フィールドが削除されています。GITOPS-1777

- 今回の更新以前は、Operator を使用して Argo CD の HA バリアントをインストールした場合、Operator は **podAntiAffinity** ルールではなく、**podAffinity** ルールで Redis **StatefulSet** オブジェクトを作成していました。今回の更新によりこの問題は修正され、Operator は **podAntiAffinity** ルールで Redis **StatefulSet** を作成するようになりました。GITOPS-1645

- 今回の更新以前は、Argo CD **ApplicationSet** で **ssh** Zombie プロセスが多すぎていました。今回の更新でこの問題が修正され、プロセスを生成してゾンビを刈り取る単純な init デーモンである tini が **ApplicationSet** コントローラーに追加されます。これにより、**SIGTERM** シグナルが実行中のプロセスに適切に渡されるようになり、zombie プロセスを防ぐことができます。GITOPS-2108

5.1.17.3. 既知の問題

- Red Hat OpenShift GitOps Operator は、Dex に加えて、OIDC を介して RHSSO (KeyCloak) を利用できます。ただし、最新のセキュリティー修正が適用されると、一部のシナリオで RHSSO の証明書は検証できません。GITOPS-2214
  回避策として、ArgoCD 仕様で OIDC (Keycloak/RHSSO) エンドポイントの TLS 検証を無効にします。

```
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
...
```

5.1.18. Red Hat OpenShift GitOps 1.5.9 のリリースノート

Red Hat OpenShift GitOps 1.5.9 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.18.1. 修正された問題

- この更新の前は、Argo CD v1.8.2 以降のすべてのバージョンは、不適切な認証バグに対して脆弱でした。その結果、Argo CD はクラスターへのアクセスを許可されていない可能性のあるユーザーのトークンを受け入れていました。この問題は修正されています。CVE-2023-22482

5.1.19. Red Hat OpenShift GitOps 1.5.7 のリリースノート

Red Hat OpenShift GitOps 1.5.7 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.19.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- OpenShift Container Platform 4.12 以降、コンソールのインストールはオプションです。この修正により、Red Hat OpenShift GitOps Operator が更新され、コンソールがインストールされていない場合に Operator でエラーが発生するのを防ぐことができます。GITOPS-2353

5.1.20. Red Hat OpenShift GitOps 1.5.6 のリリースノート

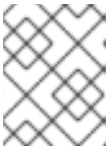
Red Hat OpenShift GitOps 1.5.6 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.20.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、ライブネスプローブが応答しないため、多数のアプリケーションでアプリケーションコントローラーが複数回再起動されていました。この更新は、アプリケーションコントローラーの **StatefulSet** オブジェクトで liveness プローブを削除することにより、問題を修正します。GITOPS-2153

- この更新の前は、証明機関によって署名されていない証明書を使用してセットアップされていると、RHSSO 証明書を検証できませんでした。今回の更新で問題が修正され、通信時に Keycloak の TLS 証明書を検証する際に使用されるカスタム証明書を提供できるようになりました。**rootCA** を Argo CD カスタムリソース **.spec.keycloak.rootCA** フィールドに追加できます。Operator はこの変更を調整し、**argocd-cm ConfigMap** の **oidc.config** フィールドを PEM エンコードされたルート証明書で更新します。GITOPS-2214

  注記

  .spec.keycloak.rootCA フィールドを更新した後、Argo CD サーバー Pod を再起動します。

  以下に例を示します。

  ```
  apiVersion: argoproj.io/v1alpha1
  kind: ArgoCD
  metadata:
    name: example-argocd
    labels:
      example: basic
  spec:
    sso:
      provider: keycloak
      keycloak:
       rootCA: |
         ---- BEGIN CERTIFICATE ----
         This is a dummy certificate
         Please place this section with appropriate rootCA
         ---- END CERTIFICATE ----
    server:
      route:
        enabled: true
  ```

- この更新の前は、Argo CD に管理されていた namespace が終了すると、ロールの作成や他の管理された namespace のその他の設定がブロックされていました。今回の更新でこの問題は修正されています。GITOPS-2278

- この更新の前は、**anyuid** の SCC が Dex **ServiceAccount** リソースに割り当てられている場合、Dex Pod は **CreateContainerConfigError** で開始できませんでした。この更新プログラムでは、デフォルトのユーザー ID を Dex コンテナーに割り当てることで、この問題を修正しています。GITOPS-2235

5.1.21. Red Hat OpenShift GitOps 1.5.5 のリリースノート

Red Hat OpenShift GitOps 1.5.5 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.21.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- この更新により、同梱の Argo CD がバージョン 2.3.7 に更新されました。

5.1.21.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、より制限的な SCC がクラスターに存在する場合、ArgoCD インスタンスの **redis-ha-haproxy** Pod が失敗していました。この更新プログラムは、ワークロードのセキュリティーコンテキストを更新することで問題を修正します。GITOPS-2034

5.1.21.3. 既知の問題

- Red Hat OpenShift GitOps Operator は、OIDC および Dex で RHSSO (KeyCloak) を使用できます。ただし、最近のセキュリティー修正が適用されているため、Operator は一部のシナリオで RHSSO 証明書を検証できません。GITOPS-2214
  回避策として、ArgoCD 仕様で OIDC (Keycloak/RHSSO) エンドポイントの TLS 検証を無効にします。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
...
```

5.1.22. Red Hat OpenShift GitOps 1.5.4 リリースノート

Red Hat OpenShift GitOps 1.5.4 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.22.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、Red Hat OpenShift GitOps は古いバージョンの **REDIS 5** イメージタグを使用していました。この更新により、問題が修正され、**rhel8/redis-5** イメージタグがアップグレードされます。GITOPS-2037

5.1.23. Red Hat OpenShift GitOps 1.5.3 のリリースノート

Red Hat OpenShift GitOps 1.5.3 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.23.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前に、Argo CD v1.0.0 以降のパッチが適用されていないすべてのバージョンは、クロスサイトスクリプティングのバグに対して脆弱でした。その結果、許可されていないユーザーが UI に JavaScript リンクを挿入できる可能性があります。この問題は修正されています。CVE-2022-31035
- この更新の前に、Argo CD v0.11.0 以降のすべてのバージョンは、Argo CD CLI または UI から SSO ログインが開始されたときに、複数の攻撃に対して脆弱でした。この問題は修正されています。CVE-2022-31034
- この更新の前に、Argo CD v0.7 以降のパッチが適用されていないすべてのバージョンは、メモリー消費のバグに対して脆弱でした。その結果、許可されていないユーザーが Argo CD のリポジトリーサーバーをクラッシュさせる可能性があります。この問題は修正されています。CVE-2022-31016
- この更新の前に、Argo CD v1.3.0 以降のパッチが適用されていないすべてのバージョンは、symlink-following バグに対して脆弱でした。その結果、リポジトリーの書き込みアクセスのある権限のないユーザーが、Argo CD の repo-server から機密の YAML ファイルを漏洩する可能性がありました。この問題は修正されています。CVE-2022-31036

5.1.24. Red Hat OpenShift GitOps 1.5.2 のリリースノート

Red Hat OpenShift GitOps 1.5.2 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.24.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、**redhat-operator-index** によって参照されるイメージがありませんでした。この問題は修正されています。GITOPS-2036

5.1.25. Red Hat OpenShift GitOps 1.5.1 のリリースノート

Red Hat OpenShift GitOps 1.5.1 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.25.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、Argo CD の匿名アクセスが有効になっている場合、認証されていないユーザーが JWT トークンを作成し、Argo CD インスタンスへのフルアクセスを取得できました。この問題は修正されています。CVE-2022-29165
- この更新の前は、認証されていないユーザーは、SSO が有効になっているときにログイン画面にエラーメッセージを表示できました。この問題は修正されています。CVE-2022-24905

- この更新の前に、Argo CD v0.7.0 以降のパッチが適用されていないすべてのバージョンは、symlink-following バグに対して脆弱でした。その結果、レポジトリへの書き込みアクセスを持つ許可されていないユーザーが、機密ファイルを Argo CD のレポサーバーから漏えいする可能性があります。この問題は修正されています。CVE-2022-24904

5.1.26. Red Hat OpenShift GitOps 1.5.0 のリリースノート

Red Hat OpenShift GitOps 1.5.0 は、OpenShift Container Platform 4.8、4.9、4.10、および 4.11 で利用できるようになりました。

5.1.26.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- 今回の機能拡張により、Argo CD がバージョン **2.3.3** にアップグレードされました。GITOPS-1708
- この拡張機能により、Dex がバージョン **2.30.3** にアップグレードされます。GITOPS-1850
- 今回の機能拡張により、Helm がバージョン **3.8.0** にアップグレードされました。GITOPS-1709
- この機能拡張により、Kustomize がバージョン **4.4.1** にアップグレードされます。GITOPS-1710
- この機能拡張により、アプリケーションセットがバージョン **0.4.1** にアップグレードされます。
- この更新では、Red Hat OpenShift GitOps の最新リリースを提供する **latest** という名前の新しいチャネルが追加されました。GitOps v1.5.0 の場合、Operator は **gitops-1.5**、**latest** チャネル、および既存の **stable** チャネルにプッシュされます。GitOps v1.6 以降、すべての最新リリースは **latest** チャネルにのみプッシュされ、**stable** チャネルにはプッシュされません。GITOPS-1791
- この更新により、新しい CSV は **olm.skipRange: '>=1.0.0 <1.5.0'** アノテーションを追加します。その結果、以前のリリースバージョンはすべてスキップされます。Operator は v1.5.0 に直接アップグレードします。GITOPS-1787
- この更新により、Operator は Red Hat Single Sign-On (RH-SSO) をバージョン v7.5.1 に更新します。これには以下の機能拡張が含まれます。
  - **kube:admin** クレデンシャルを含む OpenShift クレデンシャルを使用して Argo CD にログインできます。
  - RH-SSO は、OpenShift グループを使用したロールベースアクセスコントロール (RBAC) 用の Argo CD インスタンスをサポートおよび設定します。
  - RH-SSO は、**HTTP_Proxy** 環境変数を尊重します。RH-SSO は、プロキシーの背後で実行されている Argo CD の SSO として使用できます。
    GITOPS-1330
- 今回の更新により、Argo CD オペランドの **.status** フィールドに新しい **.host** フィールドが追加されました。ルートまたは入力がルートに優先順位を付けて有効になっている場合、新しい URL フィールドにルートが表示されます。ルートまたは入力から URL が提供されていない場合、**.host** フィールドは表示されません。
  ルートまたは入力が設定されているが、対応するコントローラーが適切に設定されておらず、**Ready** 状態にないか、その URL を伝播しない場合、オペランドの **.status.host** フィール

ドの値は、URL を表示する代わりに **Pending** と表示します。これは、**Available** ではなく **Pending** にすることで、オペランドの全体的なステータスに影響します。GITOPS-654

5.1.26.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、**AppProjects** に固有の RBAC ルールでは、ロールのサブジェクトフィールドにコンマを使用できないため、LDAP アカウントへのバインドが防止されていました。この更新により問題が修正され、**AppProject** 固有の RBAC ルールで複雑なロールバインディングを指定できるようになりました。GITOPS-1771

- この更新の前は、**DeploymentConfig** リソースが **0** にスケーリングされると、Argo CD は、**"replication controller is waiting for pods to run"**という可用性ステータスメッセージとともに **progressing** の状態でリソースを表示しました。この更新により、エッジケースが修正され、可用性チェックで **DeploymentConfig** リソースの正しい可用性ステータスが報告されるようになりました。GITOPS-1738

- この更新の前に、**argocd-tls-certs-cm** 設定マップの TLS 証明書は、証明書が **ArgoCD** CR 仕様の **tls.initialCerts** フィールドで設定されていない限り、Red Hat OpenShift GitOps によって削除されていました。この問題は修正されています。GITOPS-1725

- この更新の前は、**managed-by** ラベルを使用して namespace を作成しているときに、新しい namespace に多くの **RoleBinding** リソースを作成していました。この更新により問題が修正され、Red Hat OpenShift GitOps は以前のバージョンで作成された無関係な **Role** および **RoleBinding** リソースを削除します。GITOPS-1550

- この更新の前は、パススルーモードのルートの TLS 証明書には CA 名がありませんでした。その結果、Firefox 94 以降はエラーコード **SEC_ERROR_BAD_DER** で Argo CD UI に接続できませんでした。今回の更新でこの問題が修正されています。**<openshift-gitops-ca>** シークレットを削除して、再作成する必要があります。次に、**<openshift-gitops-tls>** シークレットを削除する必要があります。Red Hat OpenShift GitOps がそれを再作成した後、Firefox から Argo CD UI に再びアクセスできます。GITOPS-1548

5.1.26.3. 既知の問題

- OpenShift クラスターで **Route** リソースの代わりに **Ingress** リソースが使用されている場合、Argo CD**.status.host** フィールドは更新されません。GITOPS-1920

5.1.27. Red Hat OpenShift GitOps 1.4.13 のリリースノート

Red Hat OpenShift GitOps 1.4.13 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.27.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- OpenShift Container Platform 4.12 以降、コンソールのインストールはオプションです。この修正により、Red Hat OpenShift GitOps Operator が更新され、コンソールがインストールされていない場合に Operator でエラーが発生するのを防ぐことができます。GITOPS-2354

5.1.28. Red Hat OpenShift GitOps 1.4.12 のリリースノート

Red Hat OpenShift GitOps 1.4.12 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.28.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、ライブネスプローブが応答しないため、多数のアプリケーションでアプリケーションコントローラーが複数回再起動されていました。この更新は、アプリケーションコントローラーの **StatefulSet** オブジェクトで liveness プローブを削除することにより、問題を修正します。GITOPS-2153

- この更新の前は、証明機関によって署名されていない証明書を使用してセットアップされていると、RHSSO 証明書を検証できませんでした。今回の更新で問題が修正され、通信時に Keycloak の TLS 証明書を検証する際に使用されるカスタム証明書を提供できるようになりました。**rootCA** を Argo CD カスタムリソース **.spec.keycloak.rootCA** フィールドに追加できます。Operator はこの変更を調整し、**argocd-cm ConfigMap** の **oidc.config** フィールドを PEM エンコードされたルート証明書で更新します。GITOPS-2214

  注記

  .spec.keycloak.rootCA フィールドを更新した後、Argo CD サーバー Pod を再起動します。

  以下に例を示します。

  ```
  apiVersion: argoproj.io/v1alpha1
  kind: ArgoCD
  metadata:
    name: example-argocd
    labels:
      example: basic
  spec:
    sso:
      provider: keycloak
      keycloak:
       rootCA: |
         ---- BEGIN CERTIFICATE ----
         This is a dummy certificate
         Please place this section with appropriate rootCA
         ---- END CERTIFICATE ----
    server:
      route:
        enabled: true
  ```

- この更新の前は、Argo CD に管理されていた namespace が終了すると、ロールの作成や他の管理された namespace のその他の設定がブロックされていました。今回の更新でこの問題は修正されています。GITOPS-2276

- この更新の前は、**anyuid** の SCC が Dex **ServiceAccount** リソースに割り当てられている場合、Dex Pod は **CreateContainerConfigError** で開始できませんでした。この更新プログラムでは、デフォルトのユーザー ID を Dex コンテナーに割り当てることで、この問題を修正しています。GITOPS-2235

5.1.29. Red Hat OpenShift GitOps 1.4.11 のリリースノート

Red Hat OpenShift GitOps 1.4.11 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.29.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- この更新により、同梱の Argo CD がバージョン 2.2.12 に更新されました。

5.1.29.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前は、より制限的な SCC がクラスターに存在する場合、ArgoCD インスタンスの **redis-ha-haproxy** Pod が失敗していました。この更新プログラムは、ワークロードのセキュリティーコンテキストを更新することで問題を修正します。GITOPS-2034

5.1.29.3. 既知の問題

- Red Hat OpenShift GitOps Operator は、OIDC および Dex で RHSSO (KeyCloak) を使用できます。ただし、最近のセキュリティー修正が適用されているため、Operator は一部のシナリオで RHSSO 証明書を検証できません。GITOPS-2214
  回避策として、ArgoCD 仕様で OIDC (Keycloak/RHSSO) エンドポイントの TLS 検証を無効にします。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
...
```

5.1.30. Red Hat OpenShift GitOps 1.4.6 のリリースノート

Red Hat OpenShift GitOps 1.4.6 は OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用可能になりました。

5.1.30.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- OpenSSL のリンクの不具合を回避するために、ベースイメージが最新バージョンに更新されています: (CVE-2022-0778)。

注記

Red Hat OpenShift GitOps 1.4 の現在のリリースをインストールし、製品ライフサイクル中にさらに更新を受け取るには、**GitOps-1.4** チャネルに切り替えます。

5.1.31. Red Hat OpenShift GitOps 1.4.5 のリリースノート

Red Hat OpenShift GitOps 1.4.5 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.31.1. 修正された問題

警告

Red Hat OpenShift GitOps v1.4.3 から Red Hat OpenShift GitOps v1.4.5 に直接アップグレードする必要があります。実稼働環境では、Red Hat OpenShift GitOps v1.4.4 を使用しないでください。Red Hat OpenShift GitOps v1.4.4 に影響のある主な問題は、Red Hat OpenShift GitOps 1.4.5 で修正されました。

以下の問題は、現在のリリースで解決されています。

- 今回の更新以前は、Argo CD Pod は **ErrImagePullBackOff** 状態のままでした。以下のエラーメッセージが表示されませんでした。

```
reason: ErrImagePull
        message: >-
          rpc error: code = Unknown desc = reading manifest
          sha256:ff4ad30752cf0d321cd6c2c6fd4490b716607ea2960558347440f2f370a586a8
          in registry.redhat.io/openshift-gitops-1/argocd-rhel8: StatusCode:
          404, <HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>
```

この問題は修正されています。GITOPS-1848

5.1.32. Red Hat OpenShift GitOps 1.4.3 のリリースノート

Red Hat OpenShift GitOps 1.4.3 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.32.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 今回の更新以前は、証明書が ArgoCD CR 仕様 **tls.initialCerts** フィールドで設定されていない限り、**argocd-tls-certs-cm** 設定マップの TLS 証明書は Red Hat OpenShift GitOps によって削除されました。今回の更新でこの問題は修正されています。GITOPS-1725

5.1.33. Red Hat OpenShift GitOps 1.4.2 のリリースノート

Red Hat OpenShift GitOps 1.4.2 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.33.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 今回の更新以前は、ルートに複数の **Ingress** が割り当てられると、**Route** リソースは **Progressing** Health ステータスのままになりました。今回の更新により、ヘルスチェックが修正され、**Route** リソースの正しいヘルスステータスを報告するようになりました。GITOPS-1751

5.1.34. Red Hat OpenShift GitOps 1.4.1 のリリースノート

Red Hat OpenShift GitOps 1.4.1 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.34.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- Red Hat OpenShift GitOps Operator v1.4.0 では、以下の CRD の **spec** から説明フィールドを削除するリグレッションが導入されました。
  - **argoproj.io_applications.yaml**
  - **argoproj.io_appprojects.yaml**
  - **argoproj.io_argocds.yaml**
    今回の更新以前は、**kubectl create** を使用して **AppProject** リソースを作成した場合、同期に失敗していました。今回の更新により、前述の CRD に欠落している説明フィールドが復元されるようになりました。GITOPS-1721

5.1.35. Red Hat OpenShift GitOps 1.4.0 のリリースノート

Red Hat OpenShift GitOps 1.4.0 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.10 で利用できるようになりました。

5.1.35.1. 新機能

現在のリリースでは、以下の改善点が追加されました。

- この機能強化により、Red Hat OpenShift GitOps Application Manager CLI (**kam**) がバージョン **0.0.41** にアップグレードされます。GITOPS-1669
- 今回の機能拡張により、Argo CD がバージョン **2.2.2** にアップグレードされました。GITOPS-1532
- 今回の機能拡張により、Helm がバージョン **3.7.1** にアップグレードされました。GITOPS-1530
- 今回の機能拡張により、**DeploymentConfig**、**Route**、および **OLM Operator** アイテムのヘルスステータスが Argo CD Dashboard および OpenShift Container Platform Web コンソールに追加されました。この情報は、アプリケーションの全体的なヘルスステータスをモニターする上で役立ちます。GITOPS-655、GITOPS-915、GITOPS-916、GITOPS-1110
- 今回の更新により、Argo CD カスタムリソースに **.spec.server.replicas** 属性および **.spec.repo.replicas** 属性をそれぞれ設定して、**argocd-server** および **argocd-repo-server** コンポーネントの必要なレプリカ数を指定できるようになりました。**argocd-server** コンポーネントの Horizontal Pod Autoscaler (HPA) を設定する場合には、Argo CD カスタムリソース属性よりも優先されます。GITOPS-1245
- 管理ユーザーとして、**argocd.argoproj.io/managed-by** ラベルを使用して Argo CD に namespace へのアクセスを許可すると、namespace-admin 権限が引き継がれます。これらの

権限は、開発チームなどの非管理者に namespace を提供する管理者にとって問題となります。なぜなら、権限によって非管理者がネットワークポリシーなどのオブジェクトを変更できるからです。
今回の更新により、管理者はすべてのマネージド namespace に共通のクラスターロールを設定できるようになりました。Argo CD アプリケーションコントローラーのロールバインディングでは、Operator は **CONTROLLER_CLUSTER_ROLE** 環境変数を参照します。Argo CD サーバーのロールバインディングでは、Operator は **SERVER_CLUSTER_ROLE** 環境変数を参照します。これらの環境変数にカスタムロールが含まれる場合、Operator はデフォルトの管理者ロールを作成しません。代わりに、すべてのマネージド namespace に既存のカスタムロールを使用します。GITOPS-1290

- 今回の更新により、OpenShift Container Platform **Developer** パースペクティブの **Environment** ページには、パフォーマンスが低下したリソースを示す破損したハートのアイコンが表示されます (ステータスが **Progressing**、**Missing**、および **Unknown** のリソースは除きます)。コンソールには、同期していないリソースを示す黄色の yield 記号のアイコンが表示されます。GITOPS-1307

5.1.35.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 今回の更新の前は、URL にパスを指定せずに、Red Hat OpenShift GitOps Application Manager CLI (**kam**) へのルートにアクセスすると、役立つ情報が何もないデフォルトページがユーザーに表示されていました。今回の更新で問題が修正され、デフォルトページに **kam** CLI のダウンロードリンクが表示されるようになりました。GITOPS-923

- 今回の更新以前は、Argo CD カスタムリソースの namespace にリソースクォータを設定すると、Red Hat SSO (RH SSO) インスタンスのセットアップが失敗する可能性がありました。今回の更新では、RH SSO デプロイメント Pod の最小リソース要求を設定することで、この問題を修正しています。GITOPS-1297

- 今回の更新以前は、**argocd-repo-server** ワークロードのログレベルを変更すると、Operator はこの設定を調整しませんでした。回避策は、デプロイメントリソースを削除して、Operator が新しいログレベルでリソースを再作成するようにすることでした。今回の更新により、ログレベルは既存の **argocd-repo-server** ワークロードに対して適切に調整されるようになりました。GITOPS-1387

- 今回の更新以前は、Operator が **argocd-secret** Secret に **.data** フィールドがない Argo CD インスタンスを管理すると、そのインスタンスの Operator がクラッシュしていました。今回の更新により問題が修正され、**.data** フィールドがない場合に Operator がクラッシュしなくなりました。代わりに、シークレットが再生成され、**gitops-operator-controller-manager** リソースが再デプロイされます。GITOPS-1402

- 今回の更新以前は、**gitopsservice** サービスには、内部オブジェクトとしてのアノテーションが付けられていました。今回の更新によりアノテーションが削除され、デフォルトの Argo CD インスタンスを更新または削除し、UI を使用してインフラストラクチャーノードで GitOps ワークロードを実行できるようになりました。GITOPS-1429

5.1.35.3. 既知の問題

現行リリースの既知の問題は以下のとおりです。

- Dex 認証プロバイダーから Keycloak プロバイダーに移行すると、Keycloak でログインの問題が発生する可能性があります。
この問題を防ぐには、移行時に Argo CD カスタムリソースから **.spec.dex** セクションを削除して Dex をアンインストールします。Dex が完全にアンインストールするまで数分待ちます。次

に、**.spec.sso.provider: keycloak** を Argo CD カスタムリソースに追加して Keycloak をインストールします。

回避策として、**.spec.sso.provider: keycloak** を削除して Keycloak をアンインストールします。次に、再インストールします。GITOPS-1450、GITOPS-1331

5.1.36. Red Hat OpenShift GitOps 1.3.7 のリリースノート

Red Hat OpenShift GitOps 1.3.7 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.6 (GA サポートに制限あり) で利用できるようになりました。

5.1.36.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- この更新の前に、OpenSSL に不具合が見つかりました。この更新では、OpenSSL の不具合を回避するために、ベースイメージを最新バージョンに更新することで問題を修正しています。(CVE-2022-0778).

注記

Red Hat OpenShift GitOps 1.3 の現在のリリースをインストールし、製品ライフサイクル中にさらに更新を受け取るには、**GitOps-1.3** チャネルに切り替えます。

5.1.37. Red Hat OpenShift GitOps 1.3.6 のリリースノート

Red Hat OpenShift GitOps 1.3.6 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.6 (GA サポートに制限あり) で利用できるようになりました。

5.1.37.1. 修正された問題

以下の問題は、現在のリリースで解決されています。

- Red Hat OpenShift GitOps では、不適切なアクセス制御により管理者の権限昇格が許可されます (CVE-2022-1025)。今回の更新でこの問題が修正されています。
- パストラバーサルの欠陥により、範囲外のファイルが漏洩する可能性があります (CVE-2022-24731)。今回の更新でこの問題が修正されています。
- パストラバーサルの欠陥と不適切なアクセス制御により、範囲外のファイルが漏洩する可能性があります (CVE-2022-24730)。今回の更新でこの問題が修正されています。

5.1.38. Red Hat OpenShift GitOps 1.3.2 のリリースノート

Red Hat OpenShift GitOps 1.3.2 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.6 (GA サポートに制限あり) で利用できるようになりました。

5.1.38.1. 新機能

以下のセクションでは、修正および安定性の面での改善点に加え、Red Hat OpenShift GitOps 1.3.2 の主な新機能について説明します。

- Argo CD をバージョン **2.1.8** にアップグレード

- Dex をバージョン 2.30.0 にアップグレード

5.1.38.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 以前のバージョンでは、**Infrastructure Features** セクションの OperatorHub UI で、**Disconnected** でフィルタリングした場合、Red Hat OpenShift GitOps Operator は 検索結果に表示されませんでした。これは、Operator の CSV ファイルに関連するアノテーションが設定されていないことが原因でした。今回の更新により、**Disconnected Cluster** アノテーションがインフラストラクチャー機能として Red Hat OpenShift GitOps Operator に追加されました。GITOPS-1539

- **Namespace-scoped** Argo CD インスタンス (例: クラスターの **All Namepsaces** にスコープされていない Argo CD インスタンス) を使用する場合、Red Hat OpenShift GitOps は管理対象の namespace の一覧を動的に維持します。これらの namespace には **argocd.argoproj.io/managed-by** ラベルが含まれます。この namespace の一覧は、**Argo CD → Settings → Clusters → "in-cluster" → NAMESPACES** のキャッシュに保存されます。今回の更新以前は、これらの namespace のいずれかを削除すると、Operator はそれを無視し、namespace はリストに残りました。この動作はクラスター設定の **CONNECTION STATE** を破損し、すべての同期の試みがエラーになりました。以下に例を示します。

  > Argo service account does not have <random_verb> on <random_resource_type> in namespace <the_namespace_you_deleted>.

  このバグは修正されています。GITOPS-1521

- 今回の更新により、Red Hat OpenShift GitOps Operator には **Deep Insights** 機能レベルのアノテーションが付けられています。GITOPS-1519

- 以前のバージョンでは、Argo CD Operator は **resource.exclusion** フィールドを独自に管理していましたが、**resource.inclusion** フィールドを無視していました。これにより、**Argo CD** CR に設定された **resource.inclusion** フィールドが **argocd-cm** 設定マップで生成できませんでした。このバグは修正されています。GITOPS-1518

5.1.39. Red Hat OpenShift GitOps 1.3.1 のリリースノート

Red Hat OpenShift GitOps 1.3.1 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.6 (GA サポートに制限あり) で利用できるようになりました。

5.1.39.1. 修正された問題

- v1.3.0 にアップグレードする場合、Operator は環境変数の順序付けられたスライスを返しません。その結果、リコンサイラーが失敗し、プロキシーの背後で実行される OpenShift Container Platform クラスターでの Argo CD Pod の再作成が頻繁に生じます。今回の更新によりこの問題を修正し、Argo CD Pod が再作成されなくなりました。GITOPS-1489

5.1.40. Red Hat OpenShift GitOps 1.3 のリリースノート

Red Hat OpenShift GitOps 1.3 は、OpenShift Container Platform 4.7、4.8、4.9、および 4.6 (GA サポートに制限あり) で利用できるようになりました。

5.1.40.1. 新機能

以下のセクションでは、修正および安定性の面での改善点に加え、Red Hat OpenShift GitOps 1.3.0 の主な新機能について説明します。

- v1.3.0 の新規インストールでは、Dex が自動的に設定されます。OpenShift または **kubeadmin** 認証情報を使用して、**openshift-gitops** namespace のデフォルトの Argo CD インスタンスにログインできます。管理者は、Operator のインストール後に Dex インストールを無効にすることができます。これにより、**openshift-gitops** namespace から Dex デプロイメントが削除されます。

- Operator によってインストールされるデフォルトの Argo CD インスタンスおよび付随するコントローラーは、単純な設定の切り替えを設定することで、クラスターのインフラストラクチャーノードで実行できるようになりました。

- Argo CD の内部通信は、TLS および OpenShift クラスター証明書を使用して保護できるようになりました。Argo CD ルートは、cert-manager などの外部証明書マネージャーの使用に加えて、OpenShift クラスター証明書を使用できるようになりました。

- コンソール 4.9 の **Developer** パースペクティブの改善された **Environments** ページを使用して、Git Ops 環境への洞察を得ます。

- OLM を使用してインストールされた **DeploymentConfig** リソース、**Route** リソース、および Operator の Argo CD のカスタムヘルスチェックにアクセスできるようになりました。

- GitOps Operator は、最新の Operator-SDK で推奨される命名規則に準拠するようになりました。

  - 接頭辞 **gitops-operator-** がすべてのリソースに追加されます。

  - サービスアカウントの名前が **gitops-operator-controller-manager** に変更されました。

5.1.40.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 以前のバージョンでは、新規 namespace が Argo CD の新規インスタンスによって管理されるように設定される場合、Operator が新規 namespace を管理するために作成する新規ロールおよびバインディングにより、すぐに **非同期** になっていました。この動作は修正されています。GITOPS-1384

5.1.40.3. 既知の問題

- Dex 認証プロバイダーから Keycloak プロバイダーに移行する際に、Keycloak でログイン問題が発生する可能性があります。GITOPS-1450
  上記の問題を防ぐには、移行時に Argo CD カスタムリソースにある **.spec.dex** セクションを削除して Dex をアンインストールします。Dex が完全にアンインストールされるまで数分待機してから、**.spec.sso.provider: keycloak** を Argo CD カスタムリソースに追加して Keycloak のインストールに進みます。

  回避策として、**.spec.sso.provider: keycloak** を削除して Keycloak をアンインストールしてから、再インストールします。

5.1.41. Red Hat OpenShift GitOps 1.2.2 のリリースノート

Red Hat OpenShift GitOps 1.2.2 を OpenShift Container Platform 4.8 でご利用いただけるようになりました。

5.1.41.1. 修正された問題

現在のリリースでは、次の問題が解決されました。

- Argo CD のすべてのバージョンは、Helm チャートで使用される任意の値を渡すことを可能にするパストラバーサルバグに対して脆弱です。今回の更新により、Helm 値ファイルを渡す際の CVE-2022-24348 gitops エラー、パストラバーサル、およびシンボリックリンクの逆参照が修正されました。GITOPS-1756

5.1.42. Red Hat OpenShift GitOps 1.2.1 のリリースノート

Red Hat OpenShift GitOps 1.2.1 を OpenShift Container Platform 4.8 でご利用いただけるようになりました。

5.1.42.1. サポート表

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

テクノロジープレビュー機能のサポート範囲

以下の表では、機能は以下のステータスでマークされています。

- **TP: テクノロジープレビュー機能**
- **GA: 一般公開機能**

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表5.2 サポート表

機能	Red Hat OpenShift GitOps 1.2.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.42.2. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 以前のバージョンでは、起動時にアプリケーションコントローラーでメモリーが大幅に急増していました。アプリケーションコントローラーのフラグ **--kubectl-parallelism-limit** は、デフォルトで 10 に設定されますが、この値は Argo CD CR 仕様に **.spec.controller.kubeParallelismLimit** の数字を指定して上書きできます。GITOPS-1255
- 最新の Triggers APIs により、**kam bootstrap** コマンドの使用時に kustomization.yaml のエントリーが重複していることが原因で、Kubernetes のビルドが失敗しました。この問題に対処するために、Pipelines および Tekton トリガーコンポーネントが v0.24.2 および v0.14.2 にそれぞれ更新されました。GITOPS-1273

- ソース namespace から Argo CD インスタンスが削除されると、永続的な RBAC ロールおよびバインディングがターゲット namespace から自動的に削除されるようになりました。GITOPS-1228

- 以前のバージョンでは、Argo CD インスタンスを namespace にデプロイする際に、Argo CD インスタンスは "managed-by" ラベルを独自の namespace に変更していました。今回の修正により、namespace のラベルが解除されると同時に、namespace に必要な RBAC ロールおよびバインディングが作成され、削除されるようになりました。GITOPS-1247

- 以前のバージョンでは、Argo CD ワークロードのデフォルトのリソース要求制限 (特に repo-server およびアプリケーションコントローラーの制限) が、非常に厳しかったことがわかりました。現在は、既存のリソースクォータが削除され、リポジトリーサーバーのデフォルトのメモリー制限が 1024M に増えました。この変更は新規インストールにのみ影響することに注意してください。既存の Argo CD インスタンスのワークロードには影響はありません。GITOPS-1274

5.1.43. Red Hat OpenShift GitOps 1.2 のリリースノート

Red Hat OpenShift GitOps 1.2 を OpenShift Container Platform 4.8 でご利用いただけるようになりました。

5.1.43.1. サポート表

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

テクノロジープレビュー機能のサポート範囲

以下の表では、機能は以下のステータスでマークされています。

- **TP: テクノロジープレビュー機能**

- **GA: 一般公開機能**

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表5.3 サポート表

機能	Red Hat OpenShift GitOps 1.2
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.43.2. 新機能

以下のセクションでは、修正および安定性の面での改善点に加え、Red Hat OpenShift GitOps 1.2 の主な新機能について説明します。

- openshift-gitops namespace への読み取りまたは書き込みアクセスがない場合、GitOps Operator で **DISABLE_DEFAULT_ARGOCD_INSTANCE** 環境変数を使用でき、値を **TRUE** に設定し、デフォルトの Argo CD インスタンスが **openshift-gitops** namespace で開始されない

ようにすることができます。

- リソース要求および制限は Argo CD ワークロードで設定されるようになりました。リソースクォータは **openshift-gitops** namespace で有効になっています。そのため、openshift-gitops namespace に手動でデプロイされる帯域外ワークロードは、リソース要求および制限で設定し、リソースクォータを増やす必要がある場合があります。

- Argo CD 認証は Red Hat SSO と統合され、クラスターの OpenShift 4 アイデンティティープロバイダーに自動的に設定されるようになりました。この機能はデフォルトで無効にされています。Red Hat SSO を有効にするには、以下に示すように **ArgoCD** CR に SSO 設定を追加します。現在、**keycloak** が唯一サポートされているプロバイダーです。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
     enabled: true
```

- ルートラベルを使用してホスト名を定義して、ルーターのシャード化をサポートするようになりました。**server** (argocd サーバー)、**grafana** ルートおよび **prometheus** ルートに対するラベルの設定のサポートが利用可能になりました。ルートにラベルを設定するには、**ArgoCD** CR のサーバーのルート設定に **labels** を追加します。

argocd サーバーにラベルを設定する ArgoCD CR YAML の例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  server:
    route:
     enabled: true
     labels:
       key1: value1
       key2: value2
```

- GitOps Operator は、ラベルを適用してターゲット namespace のリソースを管理するために Argo CD インスタンスへのパーミッションを自動的に付与するようになりました。ユーザーは、ターゲット namespace に **argocd.argoproj.io/managed-by: <source-namespace>** のラベルを付けます。**source-namespace** は、argocd インスタンスがデプロイされる namespace に置き換えます。

5.1.43.3. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 以前のバージョンでは、ユーザーが openshift-gitops namespace のデフォルトのクラスターインスタンスで管理される Argo CD の追加のインスタンスを作成した場合は、新規の Argo CD インスタンスに対応するアプリケーションが **OutOfSync** ステータスのままになる可能性がありました。この問題は、所有者の参照をクラスターシークレットに追加することで解決されています。GITOPS-1025

5.1.43.4. 既知の問題

これらは Red Hat OpenShift GitOps 1.2 の既知の問題です。

- Argo CD インスタンスがソース namespace から削除されると、ターゲット namespace の **argocd.argoproj.io/managed-by** ラベルは削除されません。GITOPS-1228

- リソースクォータが Red Hat OpenShift GitOps 1.2 の openshift-gitops namespace で有効になっています。これは、手動でデプロイされる帯域外ワークロードおよび **openshift-gitops** namespace のデフォルトの Argo CD インスタンスによってデプロイされるワークロードに影響を及ぼします。Red Hat OpenShift GitOps **v1.1.2** から **v1.2** にアップグレードする場合は、このようなワークロードをリソース要求および制限で設定する必要があります。追加のワークロードがある場合は、openshift-gitops namespace のリソースクォータを増やす必要があります。
  openshift-gitops namespace の現在のリソースクォータ。

リソース	要求	制限
CPU	6688m	13750m
メモリー	4544Mi	9070Mi

  以下のコマンドを使用して CPU 制限を更新できます。

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type='json' -p='[{"op": "replace", "path": "/spec/hard/limits.cpu", "value":"9000m"}]'
```

  以下のコマンドを使用して CPU 要求を更新できます。

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type='json' -p='[{"op": "replace", "path": "/spec/hard/cpu", "value":"7000m"}]
```

  上記のコマンドのパスは、**cpu** から **memory** を置き換えてメモリーを更新できます。

5.1.44. Red Hat OpenShift GitOps 1.1 のリリースノート

Red Hat OpenShift GitOps 1.1 を OpenShift Container Platform 4.7 でご利用いただけるようになりました。

5.1.44.1. サポート表

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。

テクノロジープレビュー機能のサポート範囲

以下の表では、機能は以下のステータスでマークされています。

- **TP: テクノロジープレビュー機能**
- **GA: 一般公開機能**

これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

表5.4 サポート表

機能	Red Hat OpenShift GitOps 1.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager CLI (**kam**)	TP

5.1.44.2. 新機能

以下のセクションでは、修正および安定性の面での改善点に加え、Red Hat OpenShift GitOps 1.1 の主な新機能について説明します。

- **ApplicationSet** 機能が追加されました (テクノロジープレビュー)。**ApplicationSet** 機能は、多数のクラスターまたはモノリポジトリー内で Argo CD アプリケーションを管理する際に、自動化およびより大きな柔軟性を可能にします。また、マルチテナント Kubernetes クラスターでセルフサービスを使用できるようにします。
- Argo CD はクラスターロギングスタックおよび OpenShift Container Platform Monitoring およびアラート機能に統合されるようになりました。
- Argo CD 認証が OpenShift Container Platform に統合されるようになりました。
- Argo CD アプリケーションコントローラーが水平的なスケーリングをサポートするようになりました。
- Argo CD Redis サーバーが高可用性 (HA) をサポートするようになりました。

5.1.44.3. 修正された問題

以下の問題は、現在のリリースで解決されています。

- 以前のバージョンでは、Red Hat OpenShift GitOps は、アクティブなグローバルプロキシー設定のあるプロキシーサーバー設定で予想通りに機能しませんでした。この問題は修正され、Argo CD は Pod の完全修飾ドメイン名 (FQDN) を使用して Red Hat OpenShift GitOps Operator によって設定され、コンポーネント間の通信を有効にできるようになりました。GITOPS-703
- Red Hat OpenShift GitOps バックエンドは、Red Hat OpenShift GitOps URL の **?ref=** クエリーパラメーターを使用して API 呼び出しを行います。以前のバージョンでは、このパラメーターは URL から読み取られず、バックエンドでは常にデフォルトの参照が考慮されました。こ

の問題は修正され、Red Hat OpenShift GitOps バックエンドは Red Hat OpenShift GitOps URL から参照クエリーパラメーターを抽出し、入力参照が指定されていない場合にのみデフォルトの参照を使用します。GITOPS-817

- 以前のバージョンでは、Red Hat OpenShift GitOps バックエンドは有効な GitLab リポジトリーを見つけることができませんでした。これは、Red Hat OpenShift GitOps バックエンドが GitLab リポジトリーの **master** ではなく、ブランチ参照として **main** の有無を確認していたためです。この問題は修正されています。GITOPS-768

- OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Environments** ページには、アプリケーションのリストおよび環境の数が表示されるようになりました。このページには、すべてのアプリケーションをリスト表示する Argo CD **Applications** ページに転送する Argo CD リンクも表示されます。Argo CD **Applications** ページには、選択したアプリケーションのみをフィルターできる **LABELS** (例: **app.kubernetes.io/name=appName**) があります。GITOPS-544

5.1.44.4. 既知の問題

これらは Red Hat OpenShift GitOps 1.1 の既知の問題です。

- Red Hat OpenShift GitOps は Helm v2 および ksonnet をサポートしません。

- Red Hat SSO (RH SSO) Operator は、非接続クラスターではサポートされません。そのため、Red Hat OpenShift GitOps Operator および RH SSO 統合は非接続クラスターではサポートされません。

- OpenShift Container Platform Web コンソールから Argo CD アプリケーションを削除すると、Argo CD アプリケーションはユーザーインターフェイスで削除されますが、デプロイメントは依然としてクラスターに残ります。回避策として、Argo CD コンソールから Argo CD アプリケーションを削除します。GITOPS-830

5.1.44.5. 互換性を破る変更

5.1.44.5.1. Red Hat OpenShift GitOps v1.0.1 からのアップグレード

Red Hat OpenShift GitOps **v1.0.1** から **v1.1** にアップグレードすると、Red Hat OpenShift GitOps Operator は **openshift-gitops** namespace で作成されたデフォルトの Argo CD インスタンスの名前を **argocd-cluster** から **openshift-gitops** に変更します。

これは互換性を破る変更であり、アップグレード前に以下の手順を手動で実行する必要があります。

1. OpenShift Container Platform Web コンソールに移動し、**openshift-gitops** namespace の **argocd-cm.yml** 設定マップファイルの内容をローカルファイルにコピーします。コンテンツの例を以下に示します。

   argocd 設定マップ YAML の例

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
   selfLink: /api/v1/namespaces/openshift-gitops/configmaps/argocd-cm
   resourceVersion: '112532'
   name: argocd-cm
   uid: f5226fbc-883d-47db-8b53-b5e363f007af
   creationTimestamp: '2021-04-16T19:24:08Z'
   ```

```
managedFields:
...
namespace: openshift-gitops
labels:
  app.kubernetes.io/managed-by: argocd-cluster
  app.kubernetes.io/name: argocd-cm
  app.kubernetes.io/part-of: argocd
data: "" (1)
admin.enabled: 'true'
statusbadge.enabled: 'false'
resource.exclusions: |
  - apiGroups:
    - tekton.dev
    clusters:
    - '*'
    kinds:
    - TaskRun
    - PipelineRun
ga.trackingid: ''
repositories: |
  - type: git
    url: https://github.com/user-name/argocd-example-apps
ga.anonymizeusers: 'false'
help.chatUrl: ''
url: >-
  https://argocd-cluster-server-openshift-gitops.apps.dev-svc-4.7-
041614.devcluster.openshift.com    "" (2)
help.chatText: ''
kustomize.buildOptions: ''
resource.inclusions: ''
repository.credentials: ''
users.anonymous.enabled: 'false'
configManagementPlugins: ''
application.instanceLabelKey: ''
```

(1) **argocd-cm.yml** 設定マップファイルの内容の **data** セクションのみを手動で復元します。

(2) 設定マップエントリーの URL の値を、新規インスタンス名 **openshift-gitops** に置き換えます。

2. デフォルトの **argocd-cluster** インスタンスを削除します。
3. 新規の **argocd-cm.yml** 設定マップファイルを編集して、**data** セクション全体を手動で復元します。
4. 設定マップエントリーの URL の値を、新規インスタンス名 **openshift-gitops** に置き換えます。たとえば、前述の例では、URL の値を以下の URL の値に置き換えます。

```
url: >-
  https://openshift-gitops-server-openshift-gitops.apps.dev-svc-4.7-
041614.devcluster.openshift.com
```

5. Argo CD クラスターにログインし、直前の設定が存在することを確認します。

5.2. OPENSHIFT GITOPS について

5.2.1. GitOps について

GitOps は、クラウドネイティブアプリケーションの継続的デプロイメントを実装するための宣言的な方法です。GitOps を使用して、複数クラスターの Kubernetes 環境全体で、OpenShift Container Platform クラスターおよびアプリケーションを管理するための反復可能なプロセスを作成できます。GitOps は、速いペースで複雑なデプロイメントを処理して自動化し、デプロイメントおよびリリースサイクルでの時間を節約します。

GitOps ワークフローは、開発、テスト、ステージング、および実稼働環境にアプリケーションをプッシュします。GitOps は新しいアプリケーションをデプロイするか、既存のアプリケーションを更新するため、必要なのはリポジトリーの更新のみとなります。他のものはすべて GitOps が自動化します。

GitOps は、Git プル要求を使用してインフラストラクチャーおよびアプリケーションの設定を管理する一連の手法で設定されます。GitOps では、Git リポジトリーが、システムおよびアプリケーション設定の信頼できる唯一の情報源 (source of truth) になります。この Git リポジトリーには、指定した環境に必要なインフラストラクチャーの宣言的な説明が含まれ、環境を説明した状態に一致させるための自動プロセスが含まれます。また、Git リポジトリーにはシステムの全体の状態が含まれるため、システムの状態への変更の追跡情報が表示され、監査可能になります。GitOps を使用することで、インフラストラクチャーおよびアプリケーション設定のスプロールの問題を解決します。

GitOps は、インフラストラクチャーおよびアプリケーションの定義をコードとして定義します。次に、このコードを使用して複数のワークスペースおよびクラスターを管理し、インフラストラクチャーおよびアプリケーション設定の作成を単純化します。コードの原則に従って、クラスターおよびアプリケーションの設定を Git リポジトリーに保存し、Git ワークフローに従って、これらのリポジトリーを選択したクラスターに適用することができます。Git リポジトリーでのソフトウェアの開発およびメンテナンスのコアとなる原則を、クラスターおよびアプリケーションの設定ファイルの作成および管理に適用できます。

5.2.2. Red Hat OpenShift GitOps について

Red Hat OpenShift GitOps は、異なる環境 (開発、ステージ、実稼働環境など) の異なるクラスターにアプリケーションをデプロイする場合に、アプリケーションの一貫性を確保します。Red Hat OpenShift GitOps は、設定リポジトリーに関連するデプロイメントプロセスを整理し、それらを中心的な要素にします。これには、少なくとも 2 つのリポジトリーが常に含まれます。

1. ソースコードを含むアプリケーションリポジトリー
2. アプリケーションの必要な状態を定義する環境設定リポジトリー

これらのリポジトリーには、指定した環境で必要なインフラストラクチャーの宣言的な説明が含まれます。また、環境を記述された状態に一致させる自動プロセスも含まれています。

Red Hat OpenShift GitOps は Argo CD を使用してクラスターリソースを維持します。Argo CD は、アプリケーションの継続的インテグレーションおよび継続的デプロイメント (CI/CD) のオープンソースの宣言型ツールです。Red Hat OpenShift GitOps は Argo CD をコントローラーとして実装し、Git リポジトリーで定義されるアプリケーション定義および設定を継続的に監視します。次に、Argo CD は、これらの設定の指定された状態をクラスターのライブ状態と比較します。

Argo CD は、指定した状態から逸脱する設定を報告します。これらの報告により、管理者は、設定を定義された状態に自動または手動で再同期することができます。したがって、ArgoCD を使用して、OpenShift Container Platform クラスターを設定するために使用されるリソースなどのグローバルカスタムリソースを配信できます。

5.2.2.1. 主な特長

Red Hat OpenShift GitOps は、以下のタスクを自動化する上で役立ちます。

- クラスターに設定、モニタリングおよびストレージについての同様の状態があることの確認。
- 複数の OpenShift Container Platform クラスターに対する設定変更を適用するか、これを元に戻す。
- テンプレート化された設定の複数の異なる環境への関連付け。
- ステージから実稼働環境へと、クラスター全体でのアプリケーションのプロモート。

5.3. RED HAT OPENSHIFT GITOPS のインストール

Red Hat OpenShift GitOps は Argo CD を使用して、クラスター Operator、オプションの Operator Lifecycle Manager (OLM) Operator、ユーザー管理など、特定のクラススコープのリソースを管理します。

以下では、Red Hat OpenShift GitOps Operator を OpenShift Container Platform クラスターにインストールし、Argo CD インスタンスにログインする方法について説明します。

重要

latest チャネルにより、Red Hat OpenShift GitOps Operator の最新の安定バージョンをインストールできます。現在、Red Hat OpenShift GitOps Operator をインストールするためのデフォルトのチャネルです。

Red Hat OpenShift GitOps Operator の特定のバージョンをインストールするには、クラスター管理者は対応する **gitops-<version>** チャネルを使用できます。たとえば、Red Hat OpenShift GitOps Operator バージョン 1.8.x をインストールするには、**gitops-1.8** チャネルを使用できます。

5.3.1. Red Hat OpenShift GitOps Operator を Web コンソールにインストールする

前提条件

- OpenShift Container Platform Web コンソールにアクセスします。
- **cluster-admin** ロールを持つアカウントがある。
- 管理者として OpenShift Container Platform クラスターにログインしている。

警告

Red Hat OpenShift GitOps Operator をインストールする前にコミュニティーバージョンの Argo CD Operator がすでにインストールされている場合は、Argo CD Community Operator 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、左側のメニューにある **Operators → OperatorHub** に移動します。
2. **OpenShift GitOps** を検索し、**Red Hat OpenShift GitOps** タイルをクリックし、**Install** をクリックします。
   Red Hat OpenShift GitOps は、クラスターのすべての namespace にインストールされます。

Red Hat OpenShift GitOps Operator がインストールされると、**openshift-gitops** namespace で利用可能なすぐに使える Argo CD インスタンスが自動的に設定され、Argo CD アイコンがコンソールツールバーに表示されます。プロジェクトでアプリケーション用に後続の Argo CD インスタンスを作成できます。

5.3.2. CLI を使用した Red Hat OpenShift GitOps Operator のインストール

CLI を使用して OperatorHub から Red Hat OpenShift GitOps 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 GitOps 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-gitops-operator
  namespace: openshift-operators
spec:
  channel: latest (1)
  installPlanApproval: Automatic
  name: openshift-gitops-operator (2)
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace (4)
```

   (1) 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 の名前を指定します。

   (3) Operator を提供する CatalogSource の名前を指定します。

   (4) CatalogSource の namespace。デフォルトの OperatorHub CatalogSource には **openshift-marketplace** を使用します。

2. **Subscription** をクラスターに適用します。

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

3. インストールが完了したら、**openshift-gitops** namespace のすべての Pod が実行されていることを確認します。

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

出力例

```
NAME                                                 READY   STATUS RESTARTS   AGE
cluster-b5798d6f9-zr576                              1/1  Running   0       65m
kam-69866d7c48-8nsjv                                 1/1  Running   0       65m
openshift-gitops-application-controller-0            1/1  Running   0       53m
openshift-gitops-applicationset-controller-6447b8dfdd-5ckgh 1/1  Running   0       65m
openshift-gitops-redis-74bd8d7d96-49bjf              1/1  Running   0       65m
openshift-gitops-repo-server-c999f75d5-l4rsg         1/1  Running   0       65m
openshift-gitops-server-5785f7668b-wj57t             1/1  Running   0       53m
```

5.3.3. Argo CD 管理アカウントを使用した Argo CD インスタンスへのログイン

Red Hat OpenShift GitOps Operator は **openshift-gitops** namespace で利用可能なすぐに使用できる Argo CD インスタンスを自動的に作成します。

前提条件

- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動し、Red Hat OpenShift GitOps Operator がインストールされていることを確認します。

2.  menu → **OpenShift GitOps → Cluster Argo CD** の順に移動します。Argo CD UI のログインページは、新規ウィンドウに表示されます。

3. オプション: OpenShift Container Platform の認証情報でログインするには、**cluster-admins** グループのユーザーであることを確認してから、Argo CD ユーザーインターフェイスで **LOG IN VIA OPENSHIFT** オプションを選択します。

注記

cluster-admins グループのユーザーになるには、**oc adm groups new cluster-admins <user>** コマンドを使用します。この場合の **<user>** は、クラスター全体またはローカルでユーザーおよびグループにバインドできるデフォルトのクラスターロールです。

4. ユーザー名とパスワードを使用してログインするには、Argo CD インスタンスのパスワードを取得します。

   a. コンソールの左側のパネルで、パースペクティブスイッチャーを使用して **Developer** パースペクティブに切り替えます。

   b. **Project** ドロップダウンリストを使用して、**openshift-gitops** プロジェクトを選択します。

   c. 左側のナビゲーションパネルを使用して、**Secrets** ページに移動します。

   d. **openshift-gitops-cluster** インスタンスを選択して、パスワードを表示します。

   e. パスワードをコピーします。

5. このパスワードおよび **admin** をユーザー名として使用し、新しいウィンドウで Argo CD UI にログインします。

注記

同じ namespace に 2 つの Argo CD CR を作成することはできません。

5.4. OPENSHIFT GITOPS のアンインストール

Red Hat OpenShift GitOps 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 GitOps Operator のデフォルト namespace に追加された Argo CD インスタンスを削除します。
2. Red Hat OpenShift GitOps 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、作成された Argo CD インスタンスは削除されません。

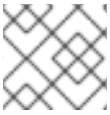
5.4.1. Argo CD インスタンスの削除

GitOps Operator の namespace に追加された Argo CD インスタンスを削除します。

手順

1. **ターミナル** に以下のコマンドを入力します。

```
$ oc delete gitopsservice cluster -n openshift-gitops
```



注記

Web コンソール UI から Argo CD クラスターを削除することはできません。

このコマンドが正常に実行されると、すべての Argo CD インスタンスは **openshift-gitops** namespace から削除されます。

同じコマンドを使用して、他の namespace から他の Argo CD インスタンスを削除します。

```
$ oc delete gitopsservice cluster -n <namespace>
```

5.4.2. GitOps 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 GitOps Operator** タイルを検索します。
2. **Red Hat OpenShift GitOps 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. **Red Hat OpenShift GitOps Operator** 記述子ページで、**Uninstall** をクリックします。

関連情報

- Operator の OpenShift Container Platform でのアンインストール方法は、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。

5.5. ARGO CD インスタンスのセットアップ

デフォルトでは、Red Hat OpenShift GitOps は Argo CD のインスタンスを **openshift-gitops** namespace にインストールし、特定のクラスタースコープのリソースを管理するための追加のアクセス許可を使用します。クラスター設定を管理したり、アプリケーションをデプロイメントしたりするために、新しい Argo CD インスタンスをインストールしてデプロイメントできます。デフォルトでは、新しいインスタンスには、デプロイされた namespace でのみリソースを管理する権限があります。

5.5.1. Argo CD のインストール

クラスター設定を管理したり、アプリケーションをデプロイメントしたりするために、新しい Argo CD インスタンスをインストールしてデプロイメントできます。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → Installed Operators** をクリックします。
3. **Project** ドロップダウンメニューから Argo CD インスタンスをインストールするプロジェクトを作成または選択します。
4. インストールした Operator から **OpenShift GitOps Operator** を選択し、**Argo CD** タブを選択します。
5. **Create** をクリックして、パラメーターを設定します。
    a. インスタンスの **Name** を入力します。デフォルトでは、**Name** は **argocd** に設定されています。
    b. 外部 OS ルートを作成して Argo CD サーバーにアクセスします。**Server → Route** をクリックし、**Enabled** にチェックを入れます。
6. Argo CD Web UI を開くには、Argo CD インスタンスがインストールされているプロジェクトで **Networking → Routes → <instance name>-server**に移動して、ルートをクリックします。

5.5.2. Argo CD サーバーとレポサーバーのレプリカを有効にする

Argo CD-server と Argo CD-repo-server のワークロードはステートレスです。ワークロードを Pod 間でより適切に分散するには、Argo CD サーバーと Argo CD リポジトリーサーバーのレプリカの数を増やすことができます。ただし、Argo CD サーバーで水平オートスケーラーが有効になっている場合は、設定したレプリカの数が上書きされます。

手順

- **repo** と **server** スペックの **replicas** パラメーターを、実行するレプリカの数に設定します。

  Argo CD カスタムリソースの例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
```

```
    name: example-argocd
    labels:
      example: repo
  spec:
    repo:
      replicas: <number_of_replicas>
    server:
      replicas: <number_of_replicas>
      route:
        enabled: true
        path: /
        tls:
          insecureEdgeTerminationPolicy: Redirect
          termination: passthrough
        wildcardPolicy: None
```

5.5.3. 別の namespace へのリソースのデプロイ

Argo CD がインストール先以外の namespace のリソースを管理できるようにするには、対象の namespace に **argocd.argoproj.io/managed-by** ラベルを設定します。

手順

- namespace を設定します。

```
$ oc label namespace <namespace> \
argocd.argoproj.io/managed-by=<instance_name> 1
```

1 Argo CD がインストールされている namespace 。

5.5.4. Argo CD コンソールリンクのカスタマイズ

マルチテナントクラスターでは、ユーザーは Argo CD の複数のインスタンスを処理する必要がある場合があります。たとえば、namespace に Argo CD インスタンスをインストールした後、コンソールアプリケーションランチャーには、独自の Argo CD インスタンスではなく、Argo CD コンソールリンクにアタッチされた別の Argo CD インスタンスが見つかる場合があります。

DISABLE_DEFAULT_ARGOCD_CONSOLELINK 環境変数を設定すると、Argo CD コンソールリンクをカスタマイズできます。

- **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** を **true** に設定すると、Argo CD コンソールリンクが完全に削除されます。
- **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** を **false** に設定するか、デフォルト値を使用すると、Argo CD コンソールリンクは、一時的に削除されますが、Argo CD ルートが調整されると、再び表示されます。

前提条件

- OpenShift Container Platform クラスターに管理者としてログインしていること。
- Red Hat OpenShift GitOps Operator がインストールされている。

手順

1. **Administrator** パースペクティブで、**Administration → CustomResourceDefinitions** に移動します。
2. **サブスクリプション** CRD を見つけて、クリックして開きます。
3. **Instances** タブを選択し、**openshift-gitops-operator** サブスクリプションをクリックします。
4. **YAML** タブを選択し、カスタマイズを行います。
   - Argo CD コンソールリンクを有効または無効にするには、必要に応じて **DISABLE_DEFAULT_ARGOCD_CONSOLELINK** の値を編集します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
spec:
  config:
    env:
    - name: DISABLE_DEFAULT_ARGOCD_CONSOLELINK
      value: 'true'
```

5.6. ARGO CD インスタンスのモニタリング

デフォルトでは、Red Hat OpenShift GitOps Operator は、定義された namespace (例: **openshift-gitops**) にインストールされている Argo CD インスタンスを自動的に検出し、これをクラスターのモニタリングスタックに接続して、非同期アプリケーションに対するアラートを提供します。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。
- 定義した namespace (たとえば **openshift-gitops**) に Argo CD アプリケーションをインストールしている。

5.6.1. Prometheus メトリクスを使用した Argo CD ヘルスのモニタリング

Prometheus メトリクスクエリーを実行して、Argo CD アプリケーションのヘルスステータスをモニタリングできます。

手順

1. Web コンソールの **Developer** パースペクティブで、Argo CD アプリケーションがインストールされている namespace を選択し、**Observe → Metrics** に移動します。
2. **Select query** ドロップダウンリストから、**Custom query** を選択します。
3. Argo CD アプリケーションのヘルスステータスを確認するには、**Expression** フィールドに、次の例のような Prometheus Query Language (PromQL) クエリーを入力します。

例

```
sum(argocd_app_info{dest_namespace=~"<your_defined_namespace>",health_status!=""})
by (health_status) 1
```

1 **<your_define_namespace>** 変数を、定義した namespace の実際の名前 (**openshift-gitops** など) に置き換えます。

5.7. クラスター設定を使用したアプリケーションのデプロイによる OPENSHIFT クラスターの設定

Red Hat OpenShift GitOps では、Argo CD を、クラスターのカスタム設定が含まれるアプリケーションと Git ディレクトリーの内容を再帰的に同期するように設定することができます。

前提条件

- OpenShift Container Platform クラスターに管理者としてログインしていること。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。
- Argo CD インスタンスにログインしました。

5.7.1. Argo CD インスタンスを使用してクラスタースコープのリソースを管理する

クラスタースコープのリソースを管理するには、Red Hat OpenShift GitOps Operator の既存の **Subscription** オブジェクトを更新し、Argo CD インスタンスの名前空間を **spec** セクションの **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 環境変数に追加します。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators → Red Hat OpenShift GitOps → Subscription** に移動します。
2. **Actions** ドロップダウンメニューをクリックし、**Edit Subscription** をクリックします。
3. **openshift-gitops-operator** サブスクリプションの詳細ページの **YAML** タブで、Argo CD インスタンスの namespace を仕様セクションの **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 環境変数に追加して、**spec** セクションの **Subscription** YAML ファイルを編集します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
...
spec:
  config:
    env:
    - name: ARGOCD_CLUSTER_CONFIG_NAMESPACES
      value: openshift-gitops, <list of namespaces of cluster-scoped Argo CD instances>
...
```

4. Argo インスタンスがクラスタースコープのリソースを管理するクラスターロールで設定されていることを確認するには、次の手順を実行します。

   a. **User Management → Roles** に移動し、**Filter** ドロップダウンメニューから **Cluster-wide Roles** を選択します。

   b. **Search by name** フィールドを使用して、**argocd-application-controller** を検索します。**Roles** ページには、作成されたクラスターロールが表示されます。

   ヒント

   あるいは、OpenShift CLI で次のコマンドを実行します。

   ```
   oc auth can-i create oauth -n openshift-gitops --as system:serviceaccount:openshift-gitops:openshift-gitops-argocd-application-controller
   ```

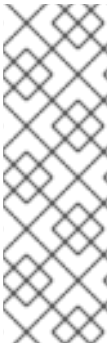
   出力 **yes** は、Argo インスタンスがクラスタースコープのリソースを管理するクラスターロールで設定されていることを確認します。それ以外の場合は、設定を確認し、必要に応じて必要な手順を実行します。

5.7.2. Argocd インスタンスのデフォルトの権限

デフォルトでは、Argo CD インスタンスには次の権限があります。

- Argo CD インスタンスには、それがデプロイされている namespace 内のリソースのみを管理する **admin** 権限があります。たとえば、**foo** namespace にデプロイされた Argo CD インスタンスには、その namespace に対してのみリソースを管理する **admin** 権限があります。
- Argo CD が適切に機能するには、リソースに対するクラスター全体の **read** 権限が必要であるため、Argo CD には次のクラスタースコープのアクセス許可があります。

```
- verbs:
  - get
  - list
  - watch
  apiGroups:
  - '*'
  resources:
  - '*'
- verbs:
  - get
  - list
  nonResourceURLs:
  - '*'
```



注記

- Argo CD が実行されている **argocd-server** と **argocd-application-controller** コンポーネントで使用されるクラスターのロールを編集して、**write** 権限が Argo CD で管理したい namespace とリソースのみに制限されるようにすることができます。

  ```
  $ oc edit clusterrole argocd-server
  $ oc edit clusterrole argocd-application-controller
  ```

5.7.3. クラスターレベルでの Argo CD インスタンスの実行

Red Hat OpenShift GitOps Operator によってインストールされるデフォルトの Argo CD インスタンスおよび付随するコントローラーは、単純な設定の切り替えを設定して、クラスターのインフラストラクチャーノードで実行できるようになりました。

手順

1. 既存のノードにラベルを付けます。

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

2. オプション: 必要な場合は、テイントを適用し、インフラストラクチャーノードでワークロードを分離し、他のワークロードがそれらのノードでスケジュールされないようにすることもできます。

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra \
infra=reserved:NoSchedule infra=reserved:NoExecute
```

3. **GitOpsService** カスタムリソースに **runOnInfra** トグルを追加します。

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. オプション: テイントがノードに追加された場合は、**tolerations** を **GitOpsService** カスタムリソースに追加します。以下に例を示します。

```
spec:
  runOnInfra: true
  tolerations:
  - effect: NoSchedule
    key: infra
    value: reserved
  - effect: NoExecute
    key: infra
    value: reserved
```

5. コンソール UI の Pod を **Pods → Pod details** で表示して、**openshift-gitops** namespace のワークロードがインフラストラクチャーノードでスケジュールされていることを確認します。

注記

デフォルトの Argo CD カスタムリソースに手動で追加された **nodeSelectors** および **tolerations** は、**GitOpsService** カスタムリソースのトグルおよび **tolerations** によって上書きされます。

関連情報

- テイントと容認の詳細は、ノードテイントを使用した Pod 配置の制御 を参照してください。

- インフラストラクチャーマシンセットの詳細は、インフラストラクチャーマシンセットの作成を参照してください。

5.7.4. Argo CD ダッシュボードを使用したアプリケーションの作成

Argo CD は、アプリケーションを作成できるダッシュボードを提供します。

このサンプルワークフローでは **cluster** ディレクトリーの内容を **cluster-configs** アプリケーションに対して再帰的に同期するために Argo CD を設定するプロセスについて説明します。ディレクトリーは Web コンソールの  メニューで Red Hat Developer Blog - **Kubernetes** へのリンクを追加する OpenShift Container Platform Web コンソールクラスター設定を定義してクラスターの namespace **spring-petclinic** を定義します。

手順

1. Argo CD ダッシュボードで、**New App** をクリックして新規の Argo CD アプリケーションを追加します。
2. このワークフローでは、以下の設定で **cluster-configs** アプリケーションを作成します。

   アプリケーション名
   cluster-configs
   プロジェクト
   default
   同期ポリシー
   Manual
   リポジトリー URL
   https://github.com/redhat-developer/openshift-gitops-getting-started
   リビジョン
   HEAD
   パス
   cluster
   宛先
   https://kubernetes.default.svc
   Namespace
   spring-petclinic
   ディレクトリーの再帰処理
   checked

3. **Create** をクリックしてアプリケーションを作成します。
4. Web コンソールの **Administrator** パースペクティブで、左側のメニューにある **Administration** → **Namespaces** に移動します。
5. namespace を検索、選択してから **Label** フィールドに **argocd.argoproj.io/managed-by=openshift-gitops** を入力し、**openshift-gitops** namespace にある Argo CD インスタンスが namespace を管理できるようにします。

5.7.5. oc ツールを使用したアプリケーションの作成

oc ツールを使用して、ターミナルで Argo CD アプリケーションを作成できます。

手順

1. サンプルアプリケーション をダウンロードします。

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. アプリケーションを作成します。

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. **oc get** コマンドを実行して、作成されたアプリケーションを確認します。

```
$ oc get application -n openshift-gitops
```

4. アプリケーションがデプロイされている namespace にラベルを追加し、**openshift-gitops** namespace の Argo CD インスタンスが管理できるようにします。

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

5.7.6. アプリケーションの Git リポジトリーとの同期

手順

1. Argo CD ダッシュボードでは、**cluster-configs** Argo CD アプリケーションに **Missing** および **OutOfSync** のステータスがあることに注意してください。アプリケーションは手動の同期ポリシーで設定されているため、Argo CD はこれを自動的に同期しません。

2. **cluster-configs** タイルの **同期** をクリックし、変更を確認してから、**Synchronize** をクリックします。Argo CD は Git リポジトリーの変更を自動的に検出します。設定が変更されると、Argo CD は **cluster-configs** のステータスを **OutOfSync** に変更します。Argo CD の同期ポリシーを変更し、Git リポジトリーからクラスターに変更を自動的に適用できるようにします。

3. **cluster-configs** Argo CD アプリケーションに **Healthy** および **Synced** のステータスがあることに注意してください。**cluster-configs** タイルをクリックし、クラスター上で同期されたリソースおよびそれらのステータスの詳細を確認します。

4. OpenShift Container Platform Web コンソールに移動し、  をクリックして **Red Hat Developer Blog - Kubernetes** へのリンクが表示されることを確認します。

5. **Project** ページに移動し、**spring-petclinic** namespace を検索し、これがクラスターに追加されていることを確認します。
クラスター設定がクラスターに正常に同期されます。

5.7.7. クラスター設定用の組み込みのアクセス許可

デフォルトでは、Argo CD インスタンスには、クラスター Operator、オプションの OLM オペレーター、およびユーザー管理など、特定のクラスタースコープのリソースを管理する権限があります。

注記

Argo CD にはクラスター管理者権限がありません。

Argo CD インスタンスのパーミッション:

Resources	説明
リソースグループ	ユーザーまたは管理者の設定
operators.coreos.com	OLM によって管理されるオプションの Operator
user.openshift.io , **rbac.authorization.k8s.io**	グループ、ユーザー、およびそれらの権限
config.openshift.io	クラスター全体のビルド設定、レジストリー設定、およびスケジューラーポリシーを設定するために使用される CVO によって管理されるコントロールプレーン Operator
storage.k8s.io	ストレージ
console.openshift.io	コンソールのカスタマイズ

5.7.8. クラスター設定のアクセス許可を追加する

Argo CD インスタンスにアクセス許可を付与して、クラスター設定を管理できます。追加のアクセス許可を持つクラスターロールを作成し、新しいクラスターロールバインディングを作成して、クラスターロールをサービスアカウントに関連付けます。

手順

1. 管理者として OpenShift Container Platform Web コンソールにログインします。
2. Web コンソールで、**User Management → Roles → Create Role** を選択します。以下の **ClusterRole** YAML テンプレートを使用してルールを追加し、追加のパーミッションを指定します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secrets-cluster-role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["*"]
```

3. **Create** をクリックしてクラスターロールを追加します。
4. ここで、クラスターのロールバインディングを作成します。Web コンソールで、**User Management → Role Bindings → Create Binding** を選択します。
5. **プロジェクト** ドロップダウンから **すべてのプロジェクト** を選択します。

6. **Create binding** をクリックします。
7. **Binding type** を **Cluster-wide role binding (ClusterRoleBinding)** として選択します。
8. **RoleBinding name** の一意の値を入力します。
9. ドロップダウンリストから、新しく作成したクラスターロールまたは既存のクラスターロールを選択します。
10. **Subject** を **ServiceAccount** として選択し、**サブジェクトの namespace** と **名前** を指定します。
    a. **Subject namespace**: **openshift-gitops**
    b. **Subject name**: **openshift-gitops-argocd-application-controller**
11. **Create** をクリックします。**ClusterRoleBinding** オブジェクトの YAML ファイルは以下のとおりです。

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-role-binding
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
```

5.7.9. Red Hat OpenShift GitOps を使用した OLM Operator のインストール

クラスター設定の Red Hat OpenShift GitOps は、特定のクラススコープのリソースを管理し、クラスター Operator または namespace スコープの OLM Operator のインストールを処理します。

クラスター管理者として、Tekton などの OLM Operator をインストールする必要がある場合を考えてみましょう。OpenShift Container Platform Web コンソールを使用して Tekton Operator を手動でインストールするか、OpenShift CLI を使用して Tekton サブスクリプションと Tekton Operator グループをクラスターに手動でインストールします。

Red Hat OpenShift GitOps は、Kubernetes リソースを Git リポジトリーに配置します。クラスター管理者は、Red Hat OpenShift GitOps を使用して、手動手順を行わずに他の OLM Operator のインストールを管理および自動化できます。たとえば、Red Hat OpenShift GitOps を使用して Tekton サブスクリプションを Git リポジトリーに配置すると、Red Hat OpenShift GitOps はこの Tekton サブスクリプションを Git リポジトリーから自動的に取得し、クラスターに Tekton Operator をインストールします。

5.7.9.1. クラススコープの Operator のインストール

Operator Lifecycle Manager (OLM) は、クラススコープの Operator の **openshift-operators** namespace 内のデフォルトの **global-operators** Operator グループを使用します。したがって、Gitops リポジトリーで **OperatorGroup** リソースを管理する必要はありません。ただし、namespace スコープの Operator の場合は、その namespace で **OperatorGroup** リソースを管理する必要があります。

クラススコープの Operator をインストールするには、必要な Operator の **Subscription** リソースを作成し、Git リポジトリーに配置します。

例: Grafana Operator サブスクリプション

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: grafana
spec:
  channel: v4
  installPlanApproval: Automatic
  name: grafana-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

5.7.9.2. namespace スコープの Operator のインストール

namespace スコープの Operator をインストールするには、必要な Operator の **Subscription** リソースと **OperatorGroup** リソースを作成して Git リポジトリーに配置します。

例: Ansible Automation Platform リソースオペレーター

```
...
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: ansible-automation-platform
...
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ansible-automation-platform-operator
  namespace: ansible-automation-platform
spec:
  targetNamespaces:
    - ansible-automation-platform
...
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ansible-automation-platform
  namespace: ansible-automation-platform
spec:
  channel: patch-me
  installPlanApproval: Automatic
  name: ansible-automation-platform-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
...
```

重要

Red Hat OpenShift GitOps を使用して複数の Operator をデプロイする場合、対応する namespace に Operator グループを 1 つだけ作成する必要があります。1 つの namespace に複数の Operator グループが存在する場合、その namespace で作成された CSV はすべて、**TooManyOperatorGroups** の理由で **failure** 状態に移行します。対応する namespace 内の Operator グループの数が 1 に達すると、以前の **failure** 状態の CSV はすべて **pending** 状態に移行します。Operator のインストールを完了するには、保留中のインストールプランを手動で承認する必要があります。

5.8. ARGO CD を使用した SPRING BOOT アプリケーションのデプロイ

Argo CD を使用すると、Argo CD ダッシュボードまたは **oc** ツールを使用して、アプリケーションを OpenShift クラスターにデプロイできます。

前提条件

- Red Hat OpenShift GitOps がクラスターにインストールされている。
- Argo CD インスタンスにログインしている。

5.8.1. Argo CD ダッシュボードを使用したアプリケーションの作成

Argo CD は、アプリケーションを作成できるダッシュボードを提供します。

このサンプルワークフローでは **cluster** ディレクトリーの内容を **cluster-configs** アプリケーションに対して再帰的に同期するために Argo CD を設定するプロセスについて説明します。ディレクトリーは Web コンソールの  メニューで Red Hat Developer Blog - **Kubernetes** へのリンクを追加する OpenShift Container Platform Web コンソールクラスター設定を定義してクラスターの namespace **spring-petclinic** を定義します。

手順

1. Argo CD ダッシュボードで、**New App** をクリックして新規の Argo CD アプリケーションを追加します。
2. このワークフローでは、以下の設定で **cluster-configs** アプリケーションを作成します。

アプリケーション名

cluster-configs

プロジェクト

default

同期ポリシー

Manual

リポジトリー URL

https://github.com/redhat-developer/openshift-gitops-getting-started

リビジョン

HEAD

パス

cluster

宛先

https://kubernetes.default.svc

Namespace

spring-petclinic

ディレクトリーの再帰処理

checked

3. このワークフローでは、以下の設定で spring-petclinic アプリケーションを作成します。

アプリケーション名

spring-petclinic

プロジェクト

default

同期ポリシー

Automatic

リポジトリー URL

https://github.com/redhat-developer/openshift-gitops-getting-started

リビジョン

HEAD

パス

app

宛先

https://kubernetes.default.svc

Namespace

spring-petclinic

4. **Create** をクリックしてアプリケーションを作成します。

5. Web コンソールの **Administrator** パースペクティブで、左側のメニューにある **Administration → Namespaces** に移動します。

6. namespace を検索、選択してから **Label** フィールドに **argocd.argoproj.io/managed-by=openshift-gitops** を入力し、**openshift-gitops** namespace にある Argo CD インスタンスが namespace を管理できるようにします。

5.8.2. oc ツールを使用したアプリケーションの作成

oc ツールを使用して、ターミナルで Argo CD アプリケーションを作成できます。

手順

1. サンプルアプリケーション をダウンロードします。

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. アプリケーションを作成します。

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. **oc get** コマンドを実行して、作成されたアプリケーションを確認します。

```
$ oc get application -n openshift-gitops
```

4. アプリケーションがデプロイされている namespace にラベルを追加し、**openshift-gitops** namespace の Argo CD インスタンスが管理できるようにします。

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

5.8.3. Argo CD の自己修復動作の確認

Argo CD は、デプロイされたアプリケーションの状態を常に監視し、Git の指定されたマニフェストとクラスターのライブの変更の違いを検出し、それらを自動的に修正します。この動作は自己修復として言及されます。

Argo CD で自己修復動作をテストし、確認することができます。

前提条件

- サンプル **app-spring-petclinic** アプリケーションがデプロイされ、設定されている。

手順

1. Argo CD ダッシュボードで、アプリケーションに **Synced** ステータスがあることを確認します。
2. Argo CD ダッシュボードの **app-spring-petclinic** タイルをクリックし、クラスターにデプロイされたアプリケーションのリソースを表示します。
3. OpenShift Container Platform Web コンソールで、**Developer** パースペクティブに移動します。
4. Spring PetClinic デプロイメントを変更し、Git リポジトリーの **app**/ ディレクトリーに変更をコミットします。Argo CD は変更をクラスターに自動的にデプロイします。
   a. OpenShift GitOps 開始のリポジトリー をフォークします。
   b. **deployment.yaml** ファイルで **failureThreshold** の値を **5** に変更します。
   c. デプロイメントクラスターで、以下のコマンドを実行し、**failureThreshold** フィールドの値を確認します。

```
$ oc edit deployment spring-petclinic -n spring-petclinic
```

5. OpenShift Container Platform Web コンソールでアプリケーションを監視している間に、クラスターでデプロイメントを変更し、これを 2 つの Pod にスケールアップして自己修復動作をテストします。
   a. 以下のコマンドを実行してデプロイメントを変更します。

```
$ oc scale deployment spring-petclinic --replicas 2  -n spring-petclinic
```

b. OpenShift Container Platform Web コンソールでは、デプロイメントは 2 つの Pod にスケールアップし、すぐに再び 1 つの Pod にスケールダウンすることに注意してください。Argo CD は Git リポジトリーとの差異を検知し、OpenShift Container Platform クラスターでアプリケーションを自動的に修復しました。

6. Argo CD ダッシュボードで、**app-spring-petclinic** タイル → **APP DETAILS** → **EVENTS** をクリックします。**EVENTS** タブには、以下のイベントが表示されます。Argo CD がクラスターのデプロイメントリソースが同期されていないことを検知し、Git リポジトリーを再同期してこれを修正します。

5.9. ARGO CD OPERATOR

ArgoCD カスタムリソースは、Argo CD クラスターを設定するコンポーネントの設定を可能にする特定の Argo CD クラスターの必要な状態を記述する Kubernetes カスタムリソース (CRD) です。

5.9.1. Argo CD CLI ツール

Argo CD CLI ツールは、コマンドラインで Argo CD を設定するのに使用されるツールです。Red Hat OpenShift GitOps は、このバイナリーをサポートしません。OpenShift コンソールを使用して Argo CD を設定します。

5.9.2. Argo CD カスタムリソースプロパティー

Argo CD カスタムリソースは以下のプロパティーで設定されます。

Name	説明	デフォルト	プロパティー
ApplicationInstanceLabelKey	Argo CD がアプリ名を追跡ラベルとして挿入する **metadata.label** キー名。	**app.kubernetes.io/instance**	

ApplicationSet	**ApplicationSet** コントローラーの設定オプション。	**<Object>**	• <Image> - **ApplicationSet** コントローラーのコンテナーイメージ。これは、**ARGOCD_APPLICATIONSET_IMAGE** 環境変数をオーバーライドします。 • <Version> - **ApplicationSet** コンテナーイメージで使用するタグ。 • <Resources> - コンテナーコンピューティングリソース。 • <LogLevel> - Argo CD Application Controller コンポーネントによって使用されるログレベル。有効なオプションは、**debug**、**info**、**error**、および **warn** です。 • <LogFormat> - Argo CD Application Controller コンポーネントが使用するログ形式。有効なオプションは **text** または **json** です。 • <PrallelismLimit> - コントローラーに設定する kubectl 並列処理の制限 **(--kubectl-parallelism-limit フラグ)**。
ConfigManagementPlugins	設定管理プラグインを追加します。	**<empty>**	

Controller	Argo CD アプリケーションコントローラーオプション。	**<Object>**	• **<Processors.Operation>** - オペレーションプロセッサーの数。 • **<Processors.Status>** - ステータスプロセッサーの数。 • **<Resources>** - コンテナーコンピューティングリソース。 • **<LogLevel>** - Argo CD Application Controller コンポーネントによって使用されるログレベル。有効なオプションは、**debug**、**info**、**error**、および **warn** です。 • **<AppSync>** - AppSync は、Argo CD アプリケーションの同期頻度を制御するために使用されます • **<Sharding.enabled>** - Argo CD Application Controller コンポーネントでシャーディングを有効にします。このプロパティーは、多数のクラスターを管理して、コントローラーコンポーネントのメモリー負荷を軽減するために使用されます。 • **<Sharding.replicas>** - Argo CD Application Controller のシャーディングをサポートするために使用されるレプリカの数。 • **<Env>** - アプリ

			ケーションコントローラーのワークロード用に設定する環境。
DisableAdmin	組み込みの管理者ユーザーを無効にします。	**false**	
GATrackingID	Google Analytics 追跡 ID を使用します。	**<empty>**	
GAAnonymizeusers	Google アナリティクスに送信されるハッシュ化されたユーザー名を有効にします。	**false**	
HA	高可用性オプション。	**<Object>**	• **<Enabled>** - Argo CD の高可用性サポートをグローバルに切り替えます。 • **<RedisProxyImage>** - Redis HAProxy コンテナーイメージ。これは、**ARGOCD_REDIS_HA_PROXY_IMAGE** 環境変数をオーバーライドします。 • **<RedisProxyVersion>** - Redis HAProxy コンテナーイメージに使用するタグ。
HelpChatURL	チャットヘルプを取得する URL(通常、これはサポート用の Slack チャネルになります)。	**https://mycorp.slack.com/argo-cd**	
HelpChatText	チャットヘルプを取得するためのテキストボックスに表示されるテキスト。	**Chat now!**	

Image	すべての Argo CD コンポーネントのコンテナーイメージ。これにより、**ARGOCD_IMAGE** 環境変数が上書きされます。	**argoproj/argocd**	
Ingress	Ingress 設定オプション。	**<Object>**	
InitialRepositories	クラスターの作成時に Argo CD を使用するように設定するための初期 Git リポジトリー。	**<empty>**	
通知	通知コントローラーの設定オプション。	**<Object>**	● **<Enabled>** - notifications-controller を開始するトグル。 ● **<Image>** - すべての Argo CD コンポーネントのコンテナーイメージ。これにより、**ARGOCD_IMAGE** 環境変数が上書きされます。 ● **<Version>** - Notifications コンテナーイメージで使用するタグ。 ● **<Resources>** - コンテナーコンピューティングリソース。 ● **<LogLevel>** - Argo CD Application Controller コンポーネントによって使用されるログレベル。有効なオプションは、**debug**、**info**、**error**、および **warn** です。

RepositoryCredentials	クラスターの作成時に Argo CD を使用するように設定するための Git リポジトリー認証情報テンプレート。	**<empty>**	
InitialSSHKnownHosts	クラスターの作成時に使用する Argo CD の SSH 既知のホストです。	**<default_Argo_CD_Known_Hosts>**	
KustomizeBuildOptions	**kustomize build** で使用するビルドオプションおよびパラメーター。	**<empty>**	
OIDCConfig	Dex の代替となる OIDC 設定。	**<empty>**	
NodePlacement	**nodeSelector** および **tolerations** を追加します。	**<empty>**	
Prometheus	Prometheus 設定オプション。	**<Object>**	• **<Enabled>** - Argo CD の Prometheus サポートをグローバルに切り替えます。 • **<Host>** - Ingress または Route リソースに使用するホスト名。 • **<Ingress>** - Prometheus の Ingress を切り替えます。 • **<Route>** - ルート設定オプション。 • **<Size>** - Prometheus **StatefulSet** のレプリカ数。

RBAC	RBAC 設定オプション。	**<Object>**	• <DefaultPolicy> - **argocd-rbac-cm 設定** マップの **policy.default** プロパティー。API リクエストを承認するときに、Argo CD がフォールバックするデフォルトのロールの名前。 • <Policy> - **argocd-rbac-cm 設定** マップの **policy.csv** プロパティー。ユーザー定義の RBAC ポリシーとロール定義を含む CSV データ。 • <Scopes> - **argocd-rbac-cm 設定** マップの scopes プロパティー。RBAC の実施中に (サブスコープに加えて) どの OIDC スコープを検査するかを制御します。

Redis	Redis 設定オプション	**<Object>**	• **<AutoTLS>** - プロバイダーを使用して、Redis サーバーの TLS 証明書 (openshift のいずれか) を作成します。現在、OpenShift Container Platform でのみ使用できます。 • **<DisableTLSVerification>** - 厳密な TLS 検証を使用して Redis サーバーにアクセスする必要があるかどうかを定義します。 • **<Image>** - Redis のコンテナーイメージ。これは、**ARGOCD_REDIS_IMAGE** 環境変数をオーバーライドします。 • **<Resources>** - コンテナーコンピューティングリソース。 • **<Version>** - Redis コンテナーイメージで使用するタグ。
ResourceCustomizations	リソースの動作をカスタマイズします。	**<empty>**	
ResourceExclusions	リソースグループのクラス全体を完全に無視します。	**<empty>**	
ResourceInclusions	適用するリソースグループ/種類を設定する設定。	**<empty>**	
Server	Argo CD Server 設定オプション。	**<Object>**	• **<Autoscale>** - サーバーの自動

			スケーリング設定オプション。 • **<ExtraCommandArgs>** - Operator によって設定された既存の引数に追加された引数のリスト。 • **<GRPC>** - GRPC 設定オプション。 • **<Host>** - Ingress または Route リソースに使用されるホスト名。 • **<Ingress>** - Argo CD サーバーコンポーネントのイングレス設定。 • **<Insecure>** - Argo CD サーバーの安全でないフラグを切り替えます。 • **<Resources>** - コンテナーコンピューティングリソース。 • **<Replicas>** - Argo CD サーバーのレプリカの数。**0** 以上である必要があります。**Autoscale** が有効になっている場合、**Replicas** は無視されます。 • **<Route>** - ルート設定オプション。 • **<Service.Type>** - サービスリソースに使用される **ServiceType**。 • **<LogLevel>** - Argo CD サーバーコンポーネントが使用する

			ログレベル。有効なオプションは、**debug**、**info**、**error**、および **warn** です。 • **<LogFormat>** - Argo CD Application Controller コンポーネントが使用するログ形式。有効なオプションは **text** または **json** です。 • **<Env>** - サーバーワークロード用に設定する環境。

SSO	シングルサインオンオプション。	**<Object>**	• **<Image>** - Keycloak のコンテナーイメージ。これにより、**ARGOCD_KEYCLOAK_IMAGE** 環境変数が上書きされます。 • **<Keycloak>** - Keycloak SSO プロバイダーの設定オプション。 • **<Dex>** - Dex SSO プロバイダーの設定オプション。 • **<Provider>** - Single Sign-On の設定に使用されるプロバイダーの名前。現在サポートされているオプションは、Dex と Keycloak です。 • **<Resources>** - コンテナーコンピューティングリソース。 • **<VerifyTLS>** - Keycloak サービスとの通信時に厳密な TLS チェックを適用するかどうか。 • **<Version>** - Keycloak コンテナーイメージで使用するタグ。
StatusBadgeEnabled	アプリケーションステータスバッジを有効にします。	**true**	

TLS	TLS 設定オプション。	**<Object>**	• **<CA.ConfigMapName>** - CA 証明書を含む **ConfigMap** の名前。 • **<CA.SecretName>** - CA 証明書とキーを含むシークレットの名前。 • **<InitialCerts>** - HTTPS 経由で Git リポジトリーに接続するための **argocd-tls-certs-cm** 設定マップ内の証明書の初期セット。
UserAnonyousEnabled	匿名ユーザーアクセスを有効にします。	**true**	
Version	すべての Argo CD コンポーネントのコンテナーイメージで使用するタグ。	最新の Argo CD バージョン	
Banner	UI バナーメッセージを追加します。	**<Object>**	• **<Banner.Content>** - バナーメッセージのコンテンツ (バナーが表示される場合に必要)。 • **<Banner.URL.SecretName>** - バナーメッセージリンクの URL (オプション)。

5.9.3. リポジトリーサーバーのプロパティー

Repo サーバーコンポーネントの設定には、次のプロパティーを使用できます。

Name	**デフォルト**	**説明**
Resources	**<empty>**	コンテナーコンピューティングリソース。

MountSAToken	**false**	**ServiceAccount** トークンを repo-server pod にマウントする必要があるかどうか。
ServiceAccount	**""**	repo-server pod で使用する **ServiceAccount** の名前。
VerifyTLS	**false**	リポジトリーサーバーとの通信時に、すべてのコンポーネントに厳密な TLS チェックを適用するかどうか。
AutoTLS	**""**	TLS のセットアップに使用するプロバイダーで、repo-server の gRPC TLS 証明書 (openshift のいずれか)。現在、OpenShift でのみ使用できます。
Image	**argoproj/argocd**	Argo CD Repo サーバーのコンテナーイメージ。これは、**ARGOCD_REPOSERVER_IMAGE** 環境変数をオーバーライドします。
Version	**.spec.Version** と同じ	Argo CD Repo サーバーで使用するタグ。
LogLevel	**info**	Argo CD Repo サーバーが使用するログレベル。有効なオプションは、debug、info、error、および warn です。
LogFormat	**text**	Argo CD Repo サーバーが使用するログ形式。有効なオプションは text または json です。
ExecTimeout	**180**	レンダリングツール (Helm、Kustomize など) の実行タイムアウト (秒単位)。
Env	**<empty>**	リポジトリーサーバーのワークロード用に設定する環境。
レプリカ	**<empty>**	Argo CD Repo サーバーのレプリカの数。**0** 以上である必要があります。

5.9.4. Argo CD インスタンスでの通知の有効化

Argo CD 通知コントローラー を有効または無効にするには、Argo CD カスタムリソースにパラメーターを設定します。デフォルトでは、通知は無効になっています。通知を有効にするには、**.yaml** ファイルで **enabled** パラメーターを **true** に設定します。

手順

1. **enabled** パラメーターを **true** に設定します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  notifications:
    enabled: true
```

5.10. REDIS との安全な通信の設定

Red Hat OpenShift GitOps で Transport Layer Security (TLS) 暗号化を使用すると、Argo CD コンポーネントと Redis キャッシュ間の通信を保護し、機密情報の可能性がある転送中のデータを保護できます。

次の設定のいずれかを使用して、Redis との通信を保護できます。

- **autotls** 設定を有効にして、TLS 暗号化に適切な証明書を発行します。
- キーと証明書のペアを使用して **argocd-operator-redis-tls** シークレットを作成し、TLS 暗号化を手動で設定します。

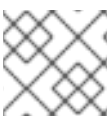
どちらの設定も、高可用性 (HA) が有効になっているかどうかに関係なく可能です。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。

5.10.1. autotls を有効にして Redis の TLS を設定する

新規または既存の Argo CD インスタンスで **autotls** 設定を有効にすることで、Redis の TLS 暗号化を設定できます。この設定では 、**argocd-operator-redis-tls** シークレットが自動的にプロビジョニングされるため、それ以上の手順は必要ありません。現時点で、OpenShift Container Platform は唯一サポートされているシークレットプロバイダーです。



注記

デフォルトでは、**autotls** 設定は無効になっています。

手順

1. OpenShift Container Platform Web コンソールにログインします。

2. **autotls** を有効にして Argo CD インスタンスを作成します。

   a. Web コンソールの **Administrator** パースペクティブで、左側のナビゲーションパネルを使用して、**Administration → CustomResourceDefinitions** に移動します。

   b. **argocds.argoproj.io** を検索し、**ArgoCD** カスタムリソース定義 (CRD) をクリックします。

   c. **CustomResourceDefinition の詳細** ページで、**Instances** タブをクリックし、**Create ArgoCD** をクリックします。

   d. 次の例のように YAML を編集または置換します。

      autotls を有効にした Argo CD CR の例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  redis:
    autotls: openshift (3)
  ha:
    enabled: true (4)
```

      (1) Argo CD インスタンスの名前。

      (2) Argo CD インスタンスを実行する namespace。

      (3) **autotls** 設定を有効にし、Redis の TLS 証明書を作成するフラグ。

      (4) HA 機能を有効にするフラグの値。HA を有効にしたくない場合は、この行を含めないか、フラグ値を **false** に設定します。

      ヒント

      あるいは、次のコマンドを実行して、既存の Argo CD インスタンスで **autotls** に設定を有効にすることもできます。

```
$ oc patch argocds.argoproj.io <instance-name> --type=merge -p '{"spec":{"redis":
{"autotls":"openshift"}}}'
```

   e. **Create** をクリックします。

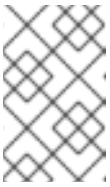
   f. Argo CD Pod が準備ができており、実行中であることを確認します。

```
$ oc get pods -n <namespace> (1)
```

      (1) Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

HA を無効にした場合の出力例

```
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0       1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm         1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq   1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh        1/1     Running   0          37s
```



注記

HA 対応の TLS 設定には、少なくとも 3 つのワーカーノードを備えたクラスターが必要です。HA 設定で Argo CD インスタンスを有効にしている場合、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

```
NAME                                          READY   STATUS    RESTARTS   AGE
argocd-application-controller-0               1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h      1/1     Running   0          10m
argocd-redis-ha-server-0                      2/2     Running   0          9m9s
argocd-redis-ha-server-1                      2/2     Running   0          98s
argocd-redis-ha-server-2                      2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh           1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks                 1/1     Running   0          10m
```

3. **argocd-operator-redis-tls** シークレットが作成されていることを確認します。

```
$ oc get secrets argocd-operator-redis-tls -n <namespace>
```
(1)

(1) Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

```
NAME                       TYPE                DATA   AGE
argocd-operator-redis-tls  kubernetes.io/tls   2      30s
```

シークレットは **kubernetes.io/tls** タイプで、サイズが **2** である必要があります。

5.10.2. autotls を無効にして Redis の TLS を設定する

キーと値のペアを使用して **argocd-operator-redis-tls** シークレットを作成して、Redis の TLS 暗号化を手動で設定できます。さらに、シークレットにアノテーションを付けて、それが適切な Argo CD インスタンスに属していることを示す必要があります。証明書とシークレットを作成する手順は、高可用性 (HA) が有効になっているインスタンスによって異なります。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. Argo CD インスタンスを作成します。

a. Web コンソールの **Administrator** パースペクティブで、左側のナビゲーションパネルを使用して、**Administration → CustomResourceDefinitions** に移動します。

b. **argocds.argoproj.io** を検索し、**ArgoCD** カスタムリソース定義 (CRD) をクリックします。

c. **CustomResourceDefinition の詳細** ページで、**Instances** タブをクリックし、**Create ArgoCD** をクリックします。

d. 次の例のように YAML を編集または置換します。

autotls を無効にした ArgoCD CR の例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: argocd (1)
  namespace: openshift-gitops (2)
spec:
  ha:
    enabled: true (3)
```

(1) Argo CD インスタンスの名前。

(2) Argo CD インスタンスを実行する namespace。

(3) HA 機能を有効にするフラグの値。HA を有効にしたくない場合は、この行を含めないか、フラグ値を **false** に設定します。

e. **Create** をクリックします。

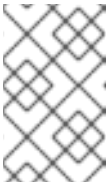
f. Argo CD Pod が準備ができており、実行中であることを確認します。

```
$ oc get pods -n <namespace> (1)
```

(1) Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

HA を無効にした場合の出力例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm       1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq 1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh      1/1     Running   0          37s
```



注記

HA 対応の TLS 設定には、少なくとも 3 つのワーカーノードを備えたクラスターが必要です。HA 設定で Argo CD インスタンスを有効にしている場合、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

```
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0           1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h   1/1     Running   0          10m
argocd-redis-ha-server-0                  2/2     Running   0          9m9s
argocd-redis-ha-server-1                  2/2     Running   0          98s
argocd-redis-ha-server-2                  2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh       1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks             1/1     Running   0          10m
```

3. HA 設定に応じて、次のいずれかのオプションを使用して、Redis サーバーの自己署名証明書を作成します。

- HA が無効になっている Argo CD インスタンスの場合は、次のコマンドを実行します。

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
  -keyout /tmp/redis.key \
  -out /tmp/redis.crt \
  -newkey rsa:4096 \
  -nodes \
  -sha256 \
  -days 10
```

1 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

```
Generating a RSA private key
..............++++
.............................++++
writing new private key to '/tmp/redis.key'
```

- HA が有効になっている Argo CD インスタンスの場合は、以下のコマンドを実行します。

```
$ openssl req -new -x509 -sha256 \
  -subj "/C=XX/ST=XX/O=Testing/CN=redis" \
  -reqexts SAN -extensions SAN \
  -config <(printf "\n[SAN]\nsubjectAltName=DNS:argocd-redis-ha-haproxy.
<namespace>.svc.cluster.local\n[req]\ndistinguished_name=req") \ 1
  -keyout /tmp/redis-ha.key \
  -out /tmp/redis-ha.crt \
  -newkey rsa:4096 \
  -nodes \
  -sha256 \
  -days 10
```

1 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

出力例

```
Generating a RSA private key
..............++++
.............................++++
writing new private key to '/tmp/redis-ha.key'
```

4. 次のコマンドを実行して、生成された証明書とキーが **/tmp** ディレクトリーで利用できることを確認します。

```
$ cd /tmp
```

```
$ ls
```

HA を無効にした場合の出力例

```
...
redis.crt
redis.key
...
```

HA を有効にした場合の出力例

```
...
redis-ha.crt
redis-ha.key
...
```

5. HA 設定に応じて、次のいずれかのオプションを使用して、**argocd-operator-redis-tls** シークレットを作成します。
   - HA が無効になっている Argo CD インスタンスの場合は、次のコマンドを実行します。

```
$ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis.key --cert=/tmp/redis.crt
```

   - HA が有効になっている Argo CD インスタンスの場合は、以下のコマンドを実行します。

```
$ oc create secret tls argocd-operator-redis-tls --key=/tmp/redis-ha.key --cert=/tmp/redis-ha.crt
```

出力例

```
secret/argocd-operator-redis-tls created
```

6. シークレットにアノテーションを付けて、それが Argo CD CR に属していることを示します。

```
$ oc annotate secret argocd-operator-redis-tls argocds.argoproj.io/name=<instance-name>
1
```

1 Argo CD インスタンスの名前を指定します (例: **argocd**)。

出力例

```
secret/argocd-operator-redis-tls annotated
```

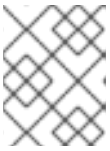
7. Argo CD Pod が準備ができており、実行中であることを確認します。

```
$ oc get pods -n <namespace> 1
```

1 Argo CD インスタンスが実行されている namespace (例: **openshift-gitops)** を指定します。

HA を無効にした場合の出力例

```
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0       1/1     Running   0          26s
argocd-redis-84b77d4f58-vp6zm         1/1     Running   0          37s
argocd-repo-server-5b959b57f4-znxjq   1/1     Running   0          37s
argocd-server-6b8787d686-wv9zh        1/1     Running   0          37s
```



注記

HA 設定で Argo CD インスタンスを有効にしている場合、出力が表示されるまでに数分かかることがあります。

HA を有効にした場合の出力例

```
NAME                                  READY   STATUS    RESTARTS   AGE
argocd-application-controller-0          1/1     Running   0          10m
argocd-redis-ha-haproxy-669757fdb7-5xg8h   1/1     Running   0          10m
argocd-redis-ha-server-0                 2/2     Running   0          9m9s
argocd-redis-ha-server-1                 2/2     Running   0          98s
argocd-redis-ha-server-2                 2/2     Running   0          53s
argocd-repo-server-576499d46d-8hgbh      1/1     Running   0          10m
argocd-server-9486f88b7-dk2ks            1/1     Running   0          10m
```

5.11. アプリケーションリソースおよびデプロイメントのヘルス情報のモニタリング

OpenShift Container Platform Web コンソールの **Developer** パースペクティブにある Red Hat OpenShift GitOps **Environments** ページには、成功したアプリケーション環境のデプロイメントのリスト、および各デプロイメントのリビジョンへのリンクが表示されます。

OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Application environments** ページには、ルート、同期ステータス、デプロイメント設定、デプロイメント履歴などのアプリケーションリソースのヘルスステータスが表示されます。

OpenShift Container Platform Web コンソールの **Developer** パースペクティブの環境ページは、Red Hat OpenShift GitOps Application Manager コマンドラインインターフェイス (CLI) の **kam** から分離さ

れています。環境が OpenShift Container Platform Web コンソールの **Developer** パースペクティブに表示されるように、**kam** を使用して、Application Environment マニフェストを生成する必要はありません。独自のマニフェストを使用できますが、環境は引き続き namespace で表す必要があります。さらに、特定のラベルとアノテーションが必要です。

5.11.1. 環境ラベルとアノテーションの設定

このセクションでは、OpenShift Container Platform Web コンソールの **Developer** パースペクティブの **Environments** ページに環境アプリケーションを表示するために必要な環境ラベルとアノテーションの設定を参考として示します。

環境ラベル
環境アプリケーションマニフェスト **には、labels.openshift.gitops/environment フィールド** と **destination.namespace** フィールドが含まれている必要があります。**<environment_name>** 変数と環境アプリケーションマニフェストの名前には、必ず同じ値を設定してください。

環境アプリケーションマニフェストの仕様

```
spec:
  labels:
    openshift.gitops/environment: <environment_name>
  destination:
    namespace: <environment_name>
...
```

環境アプリケーションマニフェストの例

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: dev-env 1
  namespace: openshift-gitops
spec:
  labels:
    openshift.gitops/environment: dev-env
  destination:
    namespace: dev-env
...
```

1 環境アプリケーションマニフェストの名前。**<environment_name>** 変数の値と同じ値を設定します。

環境アノテーション
環境 namespace マニフェストには、アプリケーションのバージョンコントローラーコードソースを指定するための **annotations.app.openshift.io/vcs-uri** フィールドと **annotations.app.openshift.io/vcs-ref** フィールドが含まれている必要があります。**<environment_name>** 変数と環境 namespace マニフェストの名前には、必ず同じ値を設定してください。

環境 namespace マニフェストの仕様

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  annotations:
    app.openshift.io/vcs-uri: <application_source_url>
    app.openshift.io/vcs-ref: <branch_reference>
  name: <environment_name> 1
...
```

1 環境 namespace マニフェストの名前。**<environment_name>** 変数の値と同じ値を設定します。

環境 namespace マニフェストの例

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    app.openshift.io/vcs-uri: https://example.com/<your_domain>/<your_gitops.git>
    app.openshift.io/vcs-ref: main
  labels:
    argocd.argoproj.io/managed-by: openshift-gitops
  name: dev-env
...
```

5.11.2. ヘルス情報の確認

Red Hat OpenShift GitOps Operator は、GitOps バックエンドサービスを **openshift-gitops** namespace にインストールします。

前提条件

- Red Hat OpenShift GitOps Operator は **OperatorHub** からインストールされます。
- アプリケーションが Argo CD によって同期されていることを確認します。

手順

1. **Developer** パースペクティブの下の **Environments** をクリックします。**Environments** ページには、**Environment status** と共にアプリケーションの一覧が表示されます。
2. **Environment status** 列の下のアイコンの上にマウスをかざすと、すべての環境の同期ステータスが表示されます。
3. リストからアプリケーション名をクリックし、特定のアプリケーションの詳細を表示します。
4. **Application environments** ページで、**Overview** タブの **Resources** セクションにアイコンが表示されている場合は、アイコンにカーソルを合わせると、ステータスの詳細が表示されます。
    - ひびの入ったハートは、リソースの問題によってアプリケーションのパフォーマンスが低下したことを示します。
    - 黄色の逆三角形は、リソースの問題により、アプリケーションのヘルスに関するデータが遅れたことを示します。

5.12. DEX を使用した ARGO CD の SSO の設定

Red Hat OpenShift GitOps Operator がインストールされると、Argo CD は **admin** パーミッションを持つユーザーを自動的に作成します。複数のユーザーを管理するために、クラスター管理者は Argo CD を使用して、シングルサインオン (SSO) を設定できます。

重要

ArgoCD CR の **spec.dex** パラメーターは非推奨です。Red Hat OpenShift GitOps v1.9 の将来のリリースでは、ArgoCD CR の **spec.dex** パラメーターを使用した Dex の設定は削除される予定です。代わりに **.spec.sso** パラメーターの使用を検討してください。

5.12.1. Dex OpenShift OAuth コネクターの有効化

Dex は、プラットフォームが提供する **OAuth** サーバーを確認して、OpenShift 内で定義されたユーザーおよびグループを使用します。以下の例は、Dex のプロパティーと設定例を紹介しています。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: openshift-oauth
spec:
  dex:
    openShiftOAuth: true 1
    groups: 2
     - default
  rbac: 3
    defaultPolicy: 'role:readonly'
    policy: |
      g, cluster-admins, role:admin
    scopes: '[groups]'
```

1. **openShiftOAuth** プロパティーは、値が **true** に設定されている場合に、組み込み OpenShift **OAuth** サーバーを自動的に設定するように Operator を トリガーします。
2. **groups** プロパティーにより、指定されたグループのユーザーはログインできます。
3. RBAC ポリシープロパティーは、Argo CD クラスターの管理者ロールを OpenShift **cluster-admins** グループのユーザーに割り当てます。

5.12.1.1. 特定のロールへのユーザーのマッピング

Argo CD は、直接の **ClusterRoleBinding** ロールがある場合は、ユーザーを特定のロールにマップできません。OpenShift 経由で SSO の **role:admin** としてロールを手動で変更できます。

手順

1. **cluster-admins** という名前のグループを作成します。

```
$ oc adm groups new cluster-admins
```

2. ユーザーをグループに追加します。

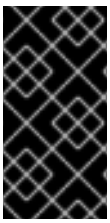
```
$ oc adm groups add-users cluster-admins USER
```

3. **cluster-admin ClusterRole** をグループに適用します。

```
$ oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
```

5.12.2. Dex の無効化

Dex は、Operator によって作成されるすべての Argo CD インスタンスにデフォルトでインストールされます。**.spec.dex** パラメーターを設定して Dex を SSO 認証プロバイダーとして使用するように Red Hat OpenShift GitOps を設定できます。



重要

Red Hat OpenShift GitOps v1.6.0 では、**DISABLE_DEX** は非推奨となり、Red Hat OpenShift GitOps v19.0 で削除される予定です。代わりに **.spec.sso.dex** パラメーターを使用することを検討してください。.spec.sso を使用した Dex の有効化または無効化を参照してください。

手順

- Operator の YAML リソースで環境変数 **DISABLE_DEX** を **true** に設定します。

```
...
spec:
  config:
    env:
    - name: DISABLE_DEX
      value: "true"
...
```

5.12.3. .spec.sso を使用した Dex の有効化または無効化

.spec.sso パラメーターを設定することで、Dex を SSO 認証プロバイダーとして使用するように Red Hat OpenShift GitOps を設定できます。

手順

1. Dex を有効にするには、Operator の YAML リソースで **.spec.sso.provider: dex** パラメーターを設定します。

```
...
spec:
  sso:
    provider: dex
    dex:
      openShiftOAuth: true
...
```

2. dex を無効にするには、Argo CD カスタムリソースから **spec.sso** 要素を削除するか、別の SSO プロバイダーを指定します。

5.13. KEYCLOAK を使用した ARGO CD の SSO の設定

Red Hat OpenShift GitOps Operator がインストールされると、Argo CD は **admin** パーミッションを持つユーザーを自動的に作成します。複数のユーザーを管理するために、クラスター管理者は Argo CD を使用して、シングルサインオン (SSO) を設定できます。

前提条件

- Red Hat SSO がクラスターにインストールされている。
- Red Hat OpenShift GitOps Operator がクラスターにインストールされます。
- Argo CD がクラスターにインストールされている。

5.13.1. Keycloak での新規クライアントの設定

Dex は、Operator によって作成されるすべての Argo CD インスタンスにデフォルトでインストールされます。ただし、Dex 設定を削除し、代わりに Keycloak を追加して OpenShift 認証情報を使用して Argo CD にログインすることができます。Keycloak は Argo CD と OpenShift 間のアイデンティティーブローカーとして機能します。

手順

Keycloak を設定するには、以下の手順に従います。

1. Argo CD カスタムリソース (CR) から **.spec.sso.dex** パラメーターを削除して Dex 設定を削除し、CR を保存します。

```
dex:
   openShiftOAuth: true
   resources:
     limits:
       cpu:
       memory:
     requests:
       cpu:
       memory:
```

2. Argo CD CR で **provider** パラメーターの値を **keycloak** に設定します。
3. 次のいずれかの手順を実行して、Keycloak を設定します。
   - 安全な接続のために、次の例に示すように **rootCA** パラメーターの値を設定します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: "<PEM-encoded-root-certificate>" 1
```

```
  server:
    route:
      enabled: true
```

1 Keycloak の TLS 証明書を検証するために使用されるカスタム証明書。

Operator は **.spec.keycloak.rootCA** パラメーターの変更を調整し、**argocd-cm** 設定マップの PEM エンコードされたルート証明書で **oidc.config** パラメーターを更新します。

- 安全でない接続の場合、**rootCA** パラメーターの値を空のままにして、以下に示すように **oidc.tls.insecure.skip.verify** パラメーターを使用します。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
  sso:
    provider: keycloak
    keycloak:
      rootCA: ""
```

注記

Keycloak インスタンスのインストールおよび実行には、2 - 3 分かかります。

5.13.2. Keycloak へのログイン

Keycloak コンソールにログインしてアイデンティティーまたはロールを管理し、さまざまなロールに割り当てられたパーミッションを定義します。

前提条件

- Dex のデフォルト設定は削除されている。
- Argo CD CR は Keycloak SSO プロバイダーを使用するように設定されている。

手順

1. ログイン用の Keycloak ルート URL を取得します。

```
$ oc -n argocd get route keycloak

NAME       HOST/PORT                                          PATH   SERVICES   PORT
TERMINATION   WILDCARD
keycloak   keycloak-default.apps.ci-ln-******.origin-ci-int-aws.dev.**.com          keycloak   <all>
reencrypt     None
```

2. 環境変数としてユーザー名とパスワードを保存する Keycloak Pod 名を取得します。

```
$ oc -n argocd get pods

NAME                  READY   STATUS        RESTARTS   AGE
keycloak-1-2sjcl      1/1     Running       0          45m
```

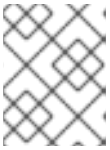
a. Keycloak ユーザー名を取得します。

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_USERNAME

SSO_ADMIN_USERNAME=<username>
```

b. Keycloak パスワードを取得します。

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_PASSWORD

SSO_ADMIN_PASSWORD=<password>
```

3. ログインページで、**LOG IN VIA KEYCLOAK** をクリックします。



注記

Keycloak インスタンスの準備ができた後にのみ、**LOGIN VIA KEYCLOAK** オプションが表示されます。

4. **Login with OpenShift** をクリックします。

注記

kubeadmin を使用したログインはサポートされていません。

5. ログインするために OpenShift の認証情報を入力します。

6. オプション: デフォルトでは、Argo CD にログインしているすべてのユーザーが、読み取り専用アクセス権を持っています。**argocd-rbac-cm** 設定マップを更新して、ユーザーレベルのアクセスを管理できます。

```
policy.csv:
<name>, <email>, role:admin
```

5.13.3. Keycloak のアンインストール

Argo CD カスタムリソース (CR) ファイルから **SSO** フィールドを削除して、Keycloak リソースおよびそれらの関連設定を削除することができます。**SSO** フィールドを削除すると、ファイルの値は以下のようになります。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
```

```
server:
  route:
   enabled: true
```

注記

この方法を使用して作成した Keycloak アプリケーションは、現在永続的ではありません。Argo CD Keycloak レルムで作成された追加の設定は、サーバーの再起動時に削除されます。

5.14. ARGO CD RBAC の設定

デフォルトでは、RHSSO を使用して Argo CD にログインする場合は、読み取り専用のユーザーになります。ユーザーレベルのアクセスを変更および管理できます。

5.14.1. ユーザーレベルのアクセス設定

ユーザーレベルのアクセスを管理および変更するには、Argo CD カスタムリソースの RBAC セクションを設定します。

手順

- **argocd** カスタムリソースを編集します。

```
$ oc edit argocd [argocd-instance-name] -n [namespace]
```

出力

```
metadata
...
...
  rbac:
    policy: 'g, rbacsystem:cluster-admins, role:admin'
    scopes: '[groups]'
```

- **policy** 設定を **rbac** セクションに追加し、**name**、**email**、およびユーザーの **role** を追加します。

```
metadata
...
...
rbac:
    policy: <name>, <email>, role:<admin>
    scopes: '[groups]'
```

注記

現在、RHSSO は Red Hat OpenShift GitOps ユーザーのグループ情報を読み取ることができません。そのため、ユーザーレベルで RBAC を設定します。

5.14.2. RHSSO リソース要求/制限の変更

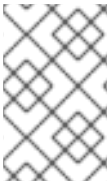
デフォルトでは、RHSSO コンテナーがリソース要求および制限と共に作成されます。リソース要求を変更および管理できます。

リソース	要求	制限
CPU	500	1000 m
メモリー	512 Mi	1024 Mi

手順

Argo CD CR のパッチを適用するデフォルトのリソース要件を変更します。

```
$ oc -n openshift-gitops patch argocd openshift-gitops --type='json' -p='[{"op": "add", "path": "/spec/sso", "value": {"provider": "keycloak", "resources": {"requests": {"cpu": "512m", "memory": "512Mi"}, "limits": {"cpu": "1024m", "memory": "1024Mi"}} }}]'
```



注記

Red Hat OpenShift GitOps によって作成された RHSSO は、Operator によって行われる変更のみを永続化します。RHSSO が再起動すると、RHSSO で Admin が作成した追加の設定が削除されます。

5.15. リソースクォータまたはリクエストの設定

Argo CD Custom Resource を使用すると、Argo CD ワークロードのリソース要求と制限を作成、更新、および削除できます。

5.15.1. リソースのリクエストと制限によるワークロードの設定

リソースの要求と制限を使用して、Argo CD カスタムリソースワークロードを作成できます。これは、リソースクォータが設定されている namespace に Argo CD インスタンスをデプロイする場合に必要です。

次の Argo CD インスタンスは、**Application Controller**、**ApplicationSet Controller**、**Dex**、**Redis**、**Repo Server**、**Server** などの Argo CD ワークロードをリソースの要求と制限とともにデプロイします。同じ方法で、リソース要件を持つ他のワークロードを作成することもできます。

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example
spec:
  server:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 125m
        memory: 128Mi
```

```
    route:
      enabled: true
  applicationSet:
    resources:
      limits:
        cpu: '2'
        memory: 1Gi
      requests:
        cpu: 250m
        memory: 512Mi
  repo:
    resources:
      limits:
        cpu: '1'
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  dex:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 250m
        memory: 128Mi
  redis:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 250m
        memory: 128Mi
  controller:
    resources:
      limits:
        cpu: '2'
        memory: 2Gi
      requests:
        cpu: 250m
        memory: 1Gi
```

5.15.2. Argo CD インスタンスにパッチを適用してリソース要件を更新する

インストール後に、すべてまたは一部のワークロードのリソース要件を更新できます。

手順

Argo CD namespace の Argo CD インスタンスの **Application Controller** リソース要求を更新します。

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/cpu", "value":"1"}]'

oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/memory", "value":"512Mi"}]'
```

5.15.3. リソース要求の削除

インストール後に、すべてまたは一部のワークロードのリソース要件を削除することもできます。

手順

Argo CD namespace の Argo CD インスタンスの **Application Controller** リソース要求を削除します。

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/cpu"}]'

oc -n argocd argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/memory"}]'
```

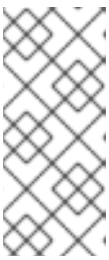
5.16. ARGO CD カスタムリソースワークロードの監視

Red Hat OpenShift GitOps を使用すると、特定の Argo CD インスタンスの Argo CD カスタムリソースワークロードの可用性を監視できます。Argo CD カスタムリソースワークロードを監視すると、Argo CD インスタンスのアラートを有効にして、その状態に関する最新情報を入手できます。対応する Argo CD インスタンスのアプリケーションコントローラー、リポジトリーサーバー、またはサーバーなどのコンポーネントワークロード Pod が特定の理由で起動できず、準備ができているレプリカの数と必要なレプリカの数の間にずれがある場合、一定期間、Operator はアラートをトリガーします。

Argo CD カスタムリソースのワークロードを監視するための設定を有効または無効にすることができます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- Red Hat OpenShift GitOps がクラスターにインストールされている。
- 監視スタックは、**openshift-monitoring** プロジェクトのクラスターで設定されます。さらに、Argo CD インスタンスは、Prometheus を介して監視できる namespace にあります。
- **kube-state-metrics** サービスがクラスターで実行されています。
- オプション: ユーザー定義プロジェクトにすでに存在する Argo CD インスタンスの監視を有効にする場合は、クラスター内の ユーザー定義プロジェクトに対して監視が有効になっている ことを確認してください。



注記

デフォルトの **openshift-monitoring** スタックによって監視されていない namespace (たとえば、**openshift-*** で始まらない namespace) で Argo CD インスタンスの監視を有効にする場合は、クラスターでユーザーワークロードの監視を有効にする必要があります。このアクションにより、監視スタックが作成された PrometheusRule を取得できるようになります。

5.16.1. Argo CD カスタムリソースワークロードの監視を有効にする

デフォルトでは、Argo CD カスタムリソースワークロードの監視設定は、**false** に設定されています。

Red Hat OpenShift GitOps を使用すると、特定の Argo CD インスタンスのワークロード監視を有効に

することができます。その結果、Operator は、特定の Argo CD インスタンスによって管理されるすべてのワークロードのアラートルールを含む **PrometheusRule** オブジェクトを作成します。これらのアラートルールは、対応するコンポーネントのレプリカ数が一定時間、望ましい状態からずれると、アラートの起動をトリガーします。Operator は、ユーザーが **PrometheusRule** オブジェクトに加えた変更を上書きしません。

手順

1. 特定の Argo CD インスタンスで **.spec.monitoring.enabled** フィールドの値を **true** に設定します。

   Argo CD カスタムリソースの例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  ...
  monitoring:
    enabled: true
  ...
```

2. Operator によって作成された PrometheusRule にアラートルールが含まれているかどうかを確認します。

   アラートルールの例

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: argocd-component-status-alert
  namespace: openshift-gitops
spec:
  groups:
    - name: ArgoCDComponentStatus
      rules:
        ...
        - alert: ApplicationSetControllerNotReady (1)
          annotations:
            message: >-
              applicationSet controller deployment for Argo CD instance in
              namespace "default" is not running
          expr: >-
            kube_statefulset_status_replicas{statefulset="openshift-gitops-application-controller
statefulset",
            namespace="openshift-gitops"} !=
            kube_statefulset_status_replicas_ready{statefulset="openshift-gitops-application-
controller statefulset",
            namespace="openshift-gitops"}
          for: 1m
          labels:
            severity: critical
```

1 Argo CD インスタンスによって作成されたワークロードが期待どおりに実行されているかどうかをチェックする PrometheusRule のアラートルール。

5.16.2. Argo CD カスタムリソースワークロードの監視の無効化

特定の Argo CD インスタンスのワークロード監視を無効にすることができます。ワークロードの監視を無効にすると、作成された PrometheusRule が削除されます。

手順

- 特定の Argo CD インスタンスで **.spec.monitoring.enabled** フィールドの値を **false** に設定します。

  Argo CD カスタムリソースの例

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  ...
  monitoring:
    enabled: false
  ...
```

5.16.3. 関連情報

- ユーザー定義プロジェクトのモニタリングの有効化

5.17. ARGO CD ログの表示

Red Hat OpenShift のロギングサブシステムを使用して Argo CD ログを表示できます。ログサブシステムは、Kibana ダッシュボード上でログを視覚化します。OpenShift Logging Operator は、デフォルトで Argo CD を使用したロギングを有効にします。

5.17.1. Argo CD ログの保存と取得

Kibana ダッシュボードを使用して、Argo CD ログを保存および取得できます。

前提条件

- Red Hat OpenShift GitOps Operator がクラスターにインストールされている。
- Red Hat OpenShift のロギングサブシステムは、デフォルト設定でクラスターにインストールされている。

手順

1. OpenShift Container Platform Web コンソールで、  メニュー → **Observability** → **Logging** に移動して Kibana ダッシュボードを表示します。
2. インデックスパターンを作成します。
   a. すべてのインデックスを表示するには、インデックスパターンを * として定義し、**Next step** をクリックします。
   b. **Time Filter field name** として **@timestamp** を選択します。
   c. **Create index pattern** をクリックします。
3. Kibana ダッシュボードのナビゲーションパネルで、**Discover** タブをクリックします。
4. Argo CD のログを取得するフィルターを作成します。次の手順では、**openshift-gitops** namespace 内のすべての Pod のログを取得するフィルターを作成します。
   a. **Add a filter +** をクリックします。
   b. **kubernetes.namespace_name** フィールドを選択します。
   c. **is** 演算子を選択します。
   d. **openshift-gitops** 値を選択します。
   e. **Save** をクリックします。
5. オプション: フィルターを追加して検索を絞り込みます。たとえば、特定の Pod のログを取得するには、フィールドとして **kubernetes.pod_name** を使用して別のフィルターを作成できます。
6. Kibana ダッシュボードでフィルタリングされた Argo CD ログを表示します。

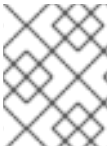
5.17.2. 関連情報

- Web コンソールを使用した Red Hat のロギングサブシステムのインストール

5.18. インフラストラクチャーノードでの GITOPS コントロールプレーンワークロードの実行

インフラストラクチャーノードを使用して、サブスクリプション数に対する追加の請求コストを防ぐことができます。

OpenShift Container Platform を使用して、Red Hat OpenShift GitOps Operator によってインストールされたインフラストラクチャーノードで特定のワークロードを実行できます。これは、デフォルトで Red Hat OpenShift GitOps Operator によって **openshift-gitops** namespace にインストールされるワークロードで設定され、その namespace のデフォルトの Argo CD インスタンスが含まれます。



注記

ユーザー namespace にインストールされたその他の Argo CD インスタンスは、インフラストラクチャーノードで実行する資格がありません。

5.18.1. GitOps ワークロードのインフラストラクチャーノードへの移行

Red Hat OpenShift GitOps によってインストールされたデフォルトのワークロードをインフラストラクチャーノードに移行できます。移動できるワークロードは以下のとおりです。

- **kam deployment**
- **cluster deployment** (バックエンドサービス)
- **openshift-gitops-applicationset-controller deployment**
- **openshift-gitops-dex-server deployment**
- **openshift-gitops-redis deployment**
- **openshift-gitops-redis-ha-haproxy deployment**
- **openshift-gitops-repo-sever deployment**
- **openshift-gitops-server deployment**
- **openshift-gitops-application-controller statefulset**
- **openshift-gitops-redis-server statefulset**

手順

1. 以下のコマンドを実行して、既存のノードにインフラストラクチャーのラベルを付けます。

```
$ oc label node <node-name> node-role.kubernetes.io/infra=
```

2. **GitOpsService** カスタムリソース (CR) を編集して、インフラストラクチャーノードセレクターを追加します。

```
$ oc edit gitopsservice -n openshift-gitops
```

3. **GitOpsService** CR ファイルで、**runOnInfra** フィールドを **spec** セクションに追加し、**true** に設定します。このフィールドは、**openshift-gitops** namespace のワークロードをインフラストラクチャーノードに移動します。

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. オプション: テイントを適用し、インフラストラクチャーノードでワークロードを分離し、他のワークロードがそれらのノードでスケジュールされないようにします。

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra
infra=reserved:NoSchedule infra=reserved:NoExecute
```

5. オプション: テイントをノードに適用する場合、容認を **GitOpsService** CR に追加できます。

```
spec:
  runOnInfra: true
```

```
tolerations:
- effect: NoSchedule
  key: infra
  value: reserved
- effect: NoExecute
  key: infra
  value: reserved
```

ワークロードが Red Hat OpenShift GitOps namespace のインフラストラクチャーノードでスケジュールされていることを確認するには、Pod 名のいずれかをクリックし、**ノードセレクター**– および **容認** が追加されていることを確認します。

注記

デフォルトの Argo CD CR の手動で追加された **ノードセレクター** および **容認** は、**GitOpsService** CR のトグルおよび容認によって上書きされます。

5.18.2. 関連情報

- テイントと容認の詳細は、ノードテイントを使用した Pod 配置の制御 を参照してください。
- インフラストラクチャーマシンセットの詳細は、インフラストラクチャーマシンセットの作成 を参照してください。

5.19. GITOPS OPERATOR のサイズ要件

サイジング要件ページには、Red Hat OpenShift GitOps に OpenShift Container Platform をインストールするためのサイジング要件が表示されます。また、GitOps オペレーターによってインスタンス化されるデフォルトの ArgoCD インスタンスのサイジングの詳細も提供します。

5.19.1. GitOps のサイジング要件

Red Hat OpenShift GitOps は、クラウドネイティブアプリケーションの継続的デプロイメントを実装するための宣言的な方法です。GitOps を使用すると、アプリケーションの CPU とメモリーの要件を定義および設定できます。

Red Hat OpenShift GitOps Operator をインストールするたびに、namespace 上のリソースが、定義された制限内でインストールされます。デフォルトのインストールで制限と要求が設定されていない場合、Operator は namespace でクォータを使用して失敗します。十分なリソースがないと、クラスターは Argo CD 関連の Pod をスケジュールできません。次の表に、デフォルトのワークロードのリソース要求および制限の詳細を示します。

ワークロード	CPU 要求	CPU 上限	メモリー要求	メモリー上限
argocd-application-controller	1	2	1024M	2048M
applicationset-controller	1	2	512M	1024M
argocd-server	0.125	0.5	128M	256M

ワークロード	CPU 要求	CPU 上限	メモリー要求	メモリー上限
argocd-repo-server	0.5	1	256M	1024M
argocd-redis	0.25	0.5	128M	256M
argocd-dex	0.25	0.5	128M	256M
HAProxy	0.25	0.5	128M	256M

オプションで、**oc** コマンドで ArgoCD カスタムリソースを使用して、詳細を確認し、変更することもできます。

```
oc edit argocd <name of argo cd> -n namespace
```

5.20. RED HAT OPENSHIFT GITOPS の問題のトラブルシューティング

Red Hat OpenShift GitOps を使用する場合、パフォーマンス、監視、設定、およびその他の側面に関連する問題に直面する場合があります。このセクションは、これらの問題を理解して解決するためのソリューションを提供するのに役立ちます。

5.20.1. 問題: Argo CD とマシン設定の同期中の自動再起動

Red Hat OpenShift Container Platform では、ノードは Red Hat OpenShift Machine Config Operator (MCO) によって自動的に更新されます。Machine Config Operator (MCO) は、クラスターがそのノードの完全なライフサイクルを管理するために使用するカスタムリソースです。

クラスターで MCO リソースが作成または更新されると、MCO は更新を取得し、選択されたノードに必要な変更を実行し、それらのノードの閉鎖、ドレイン、および再起動によってノードを正常に再起動します。カーネルから kubelet まですべてを処理します。

ただし、MCO と GitOps ワークフローの間の相互作用により、主要なパフォーマンスの問題やその他の望ましくない動作が発生する可能性があります。このセクションでは、MCO と Argo CD GitOps オーケストレーションツールをうまく連携させる方法を示します。

5.20.1.1. 解決策: マシン設定と Argo CD のパフォーマンスを向上させる

GitOps ワークフローの一部として Machine Config Operator を使用している場合、次のシーケンスではパフォーマンスが最適化されない可能性があります。

- Argo CD は、アプリケーションリソースを含む Git リポジトリーにコミットした後、自動同期ジョブを開始します。
- 同期操作の進行中に Argo CD が新しいマシン設定または更新されたマシン設定を認識すると、MCO はマシン設定への変更を取得し、ノードの再起動を開始して変更を適用します。
- クラスター内の再起動ノードに Argo CD アプリケーションコントローラーが含まれている場合、アプリケーションコントローラーが終了し、アプリケーションの同期が中止されます。

MCO はノードを順番に再起動し、再起動のたびに Argo CD ワークロードを再スケジュールできるため、同期が完了するまでに時間がかかる場合があります。これにより、MCO が同期内のマシン設定の影響を受けるすべてのノードを再起動するまで、未定義の動作が発生します。

5.20.2. 関連情報

- Argo CD とマシン設定の同期中にノードが自動再起動しないようにする