Red Hat OpenShift GitOps 1.13

Argo Rollouts

Argo Rollouts を使用したプログレッシブ配信

Last Updated: 2024-10-10

Red Hat OpenShift GitOps 1.13 Argo Rollouts

Argo Rollouts を使用したプログレッシブ配信

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

このドキュメントでは、Argo Rollouts を使用して、宣言的ロールアウト戦略に必要なすべての定義をカプセル化する手順を説明します。また、GitOps ワークフローの一部としてデプロイメントのプログレッシブ配信を使用、管理および自動化する方法についても説明します。

目次

第1章 ARGO ROLLOUTS の概要

GitOps のコンテキストでは、プログレッシブ配信は、制御された段階的な方法でアプリケーションの更新をリリースするプロセスです。プログレッシブ配信では、アプリケーション更新の新しいバージョンを最初は一部のユーザーにのみ公開することで、リリースのリスクを軽減します。このプロセスには、この新しいアプリケーションバージョンを継続的に観察および分析して、その動作が設定された要件および期待と一致するかどうかを検証することが含まれます。このプロセスによりてアプリケーションの更新が徐々に幅広いユーザーに公開されるにつれて、検証は継続されます。

OpenShift Container Platform は、ルートを使用して異なるサービス間でトラフィックを分割することにより、ある程度の段階的な配信機能を提供しますが、これには通常、手動による介入と管理が必要です。

Argo Rollouts を使用すると、クラスター管理者はプログレッシブデプロイメントの配信を自動化し、Kubernetes および OpenShift Container Platform クラスターでホストされているアプリケーションのプログレッシブデプロイメントを管理できます。Argo Rollouts は、ブルーグリーン、カナリア、カナリア分析、実験などの高度なデプロイメント機能を提供するカスタムリソース定義 (CRD) を備えたコントローラーです。

1.1. ARGO ROLLOUTS を使用する理由

クラスター管理者として、従来のインフラストラクチャーにおける高度なデプロイメントストラテジーの管理と調整には、メンテナンス期間が長くなる可能性があります。OpenShift Container Platform や Red Hat OpenShift GitOps などのツールを使用した自動化により、これらの期間を短縮できますが、これらの戦略を設定するのは依然として困難な場合があります。

Argo Rollouts を使用すると、アプリケーションチームがロールアウトストラテジーを宣言的に定義できるようになり、プログレッシブ配信が簡素化されます。チームは、複数のデプロイメントやサービスを定義したり、トラフィックシェーピングやテストの統合のための自動化を作成したりする必要がなくなりました。

以下を理由に、Argo Rollouts を使用できます。

- エンドユーザー環境でのプログレッシブ配信をより簡単に導入できます。
- Argo Rollouts の構造とガイドラインを利用すれば、チームはトラフィックマネージャーや複雑なインフラストラクチャーについて学習する必要がなくなります。
- 更新中、デプロイメントストラテジーに応じて、トラフィックを新しいバージョンに徐々に移行することで、デプロイされたアプリケーションバージョンの既存のトラフィックシェーピング機能を最適化できます。
- Argo Rollouts を Prometheus などのメトリックプロバイダーと組み合わせて、パラメーターセットに基づいてメトリックベースおよびポリシー駆動のロールアウトとロールバックを実行できます。
- エンドユーザー環境は、Red Hat OpenShift GitOps Operator のセキュリティーを取得し、リソース、コスト、および時間を効果的に管理するのに役立ちます。
- セキュリティーと自動デプロイメントを備えた Argo CD を使用している既存のユーザーは、プロセスの早い段階でフィードバックを受け取り、それを使用して影響のある問題を回避できます。

1.1.1. Argo Rollouts の利点

Argo Rollouts を Red Hat OpenShift GitOps のデフォルトのワークロードとして使用すると、次の利点があります。

- GitOps ワークフローの一部として自動化されたプログレッシブ配信
- 高度なデプロイメント機能
- Blue-Green や Canary などの既存の高度なデプロイメント戦略を最適化
- デプロイメント時のダウンタイムがない更新
- きめ細かく重み付けされたトラフィックのシフト
- 実稼働環境に新しいトラフィックが到達することなくテスト可能
- 自動化されたロールバックとプロモーション
- 手動判定
- カスタマイズ可能な指標クエリーとビジネス主要業績評価指標 (KPI) の分析
- 高度なトラフィックルーティングのための Ingress コントローラーおよび Red Hat OpenShift Service Mesh との統合
- デプロイメント戦略分析のためのメトリックプロバイダーとの統合
- 複数のプロバイダーの使用

1.2. ROLLOUTMANAGER のカスタムリソースと仕様について

Argo Rollouts を使用するには、クラスターに Red Hat OpenShift GitOps Operator をインストールし、選択した namespace で **RolloutManager** カスタムリソース (CR) を作成して Operator に送信する必要があります。**RolloutManager** CR のスコープを 1 つまたは複数の namespace に設定できます。Operator は、次の namespace スコープのサポートリソースを使用して **argo-rollouts** インスタンスを作成します。

- Argo Rollouts コントローラー
- Argo Rollouts メトリックサービス
- Argo Rollouts サービスアカウント
- Argo Rollouts のロール
- Argo Rollouts のロールバインディング
- Argo Rollouts のシークレット

RolloutsManager CR の仕様で、Argo Rollouts コントローラーリソースのコマンド引数、環境変数、カスタムイメージ名などを指定できます。**RolloutManager** CR 仕様は、Argo Rollouts の望ましい状態を定義します。

例: RolloutManager CR

```
apiVersion: argoproj.io/v1alpha1
kind: RolloutManager
metadata:
```

```
  name: argo-rollout
  labels:
    example: basic
spec: {}
```

1.2.1. Argo Rollouts コントローラー

Argo Rollouts コントローラーリソースを使用すると、namespace でのプログレッシブアプリケーション配信を管理できます。Argo Rollouts コントローラーリソースはクラスターのイベントを監視し、Argo Rollouts に関連するリソースに変更があるたびに反応します。コントローラーはロールアウトの詳細をすべて読み取り、クラスターをロールアウト定義に記述されているのと同じ状態にします。

1.3. ARGO ROLLOUTS アーキテクチャーの概要

Argo Rollouts サポートは、Red Hat OpenShift GitOps Operator をインストールし、**RolloutManager** カスタムリソース (CR) インスタンスを設定することで、クラスター上で有効になります。

RolloutManager CR が作成されると、Red Hat OpenShift GitOps Operator は同じ namespace に Argo Rollouts をインストールします。この手順には、Argo Rollouts コントローラーのインストール、および CR、ロール、ロールバインディング、設定データなどの Argo Rollouts の処理に必要なリソースが含まれます。

Argo Rollouts コントローラーは、次の 2 つのモードでインストールできます。

- **クラススコープモード** (デフォルト): コントローラーは、クラスター内のすべての namespace 全体のリソースを監視します。
- **namespace スコープモード**: コントローラーは、Argo Rollouts がデプロイされている namespace 内のリソースを監視します。

Argo Rollouts のアーキテクチャーは、コンポーネントとリソースで設定されています。コンポーネントは、リソースの管理に使用されます。たとえば、AnalysisRun コントローラーは **AnalysisRun** CR を管理します。

Argo Rollouts には、新しいアプリケーションバージョンがデプロイされていることを確認するための分析メトリクスを収集するいくつかのメカニズムが含まれています。

- **Prometheus メトリクス**: **AnalysisTemplate** CR は、Prometheus インスタンスに接続して、1 つ以上のメトリクスの成功または失敗を評価するように設定されています。
- **Kubernetes ジョブメトリクス**: Argo Rollouts は、リソースメトリクスの分析を実行するための Kubernetes **Job** リソースをサポートします。Kubernetes ジョブの実行が成功したかどうかに基づいて、アプリケーションのデプロイメントが成功したかどうかを確認できます。

1.3.1. Argo Rollouts コンポーネント

Argo Rollouts は、ユーザーが OpenShift Container Platform でプログレッシブ配信を実践できるようにする複数のコンポーネントで構成されています。

表1.1 Argo Rollouts コンポーネント

名前	説明

名前	説明
Argo Rollouts Controller	Argo Rollouts Controller は標準の **Deployment** リソースの代わりで、それと共存します。このコントローラーは、Argo Rollouts リソースの変更にのみ応答し、**Rollout** CR を管理します。Argo Rollouts Controller は標準のデプロイメントリソースを変更しません。
AnalysisRun Controller	AnalysisRun コントローラーは、**AnalysisRun** および **AnalysisTemplate** CR の分析を管理および実行します。ロールアウトをメトリクスプロバイダーに接続し、アプリケーションのデプロイメント更新が成功したかどうかを判断するメトリクスのしきい値を定義します。
Experiment controller	**Experiment** コントローラーは、有効期限の短いレプリカセットで分析を実行し、**Experiment** カスタムリソースを管理します。カナリアデプロイメント **strategy** フィールドで **experiment** ステップを指定することにより、コントローラーを **Rollout** リソースと統合することもできます。
Service および **Ingress** コントローラー	Service コントローラーは **Service** リソースを管理し、Ingress コントローラーは Argo Rollouts によって変更された **Ingress** リソースを管理します。これらのコントローラーは、トラフィック管理のためにアプリケーションインスタンスに追加のメタデータアノテーションを挿入します。
Argo Rollouts CLI と UI	Argo Rollouts は、Argo Rollouts CLI と呼ばれる **oc/kubectl** プラグインをサポートしています。これを使用して、コマンドラインからロールアウト、分析、実験などのリソースと対話できます。**pause**、**promote**、**retry** などの操作を実行できます。Argo Rollouts CLI プラグインを使用すると、ブラウザーでローカル Web UI ダッシュボードを起動して、Argo Rollouts リソースの視覚化エクスペリエンスを強化できます。

1.3.2. Argo Rollouts のリソース

Argo Rollout コンポーネントは、複数のリソースを管理して、プログレッシブ配信を有効にします。

- **Rollouts 固有のリソース**: **Rollout**、**AnalysisRun**、**Experiment** など。
- **Kubernetes ネットワークリソース**: ネットワークトラフィックシェーピングの **Service**、**Ingress**、または **Route** など。Argo Rollouts は、トラフィック管理と呼ばれるこれらのリソースと統合します。

これらのリソースは、**Rollout** CR を介してアプリケーションのデプロイメントをカスタマイズするために不可欠です。

Argo Rollouts は、次のアクションをサポートしています。

- カナリアデプロイメントのパーセンテージベースのトラフィックをルーティングする。
- **Service** および **Ingress** リソースを使用して、受信ユーザートラフィックを正しいアプリケーションバージョンに転送します。
- 複数のメカニズムを使用して分析メトリクスを収集し、アプリケーションの新しいバージョンのデプロイメントを検証します。

表1.2 Argo Rollouts のリソース

名前	説明
Rollout	この CR により、カナリアまたは blue-green デプロイメントストラテジーを使用したアプリケーションのデプロイメントが可能になります。これは、組み込みの Kubernetes **Deployment** リソースを置き換えます。
AnalysisRun	この CR は、分析を実行し、分析結果を集約して、アプリケーションの正常なデプロイメント配信に向けてユーザーを導くために使用されます。**AnalysisRun** CR は、**AnalysisTemplate** CR のインスタンスです。
AnalysisTemplate	**AnalysisTemplate** CR は、メトリクスのクエリー方法の説明を提供するテンプレートファイルです。これらの命令の結果は、**AnalysisRun** CR の形式でロールアウトに割り当てられます。**AnalysisTemplate** CR はクラスターまたは特定のロールアウトでグローバルに定義できます。**Experiment** カスタムリソースを作成すると、レプリカセットで使用される **AnalysisTemplate** のリストをリンクできます。
Experiment	**Experiment** CR は、アプリケーションが正しくデプロイされるように、デプロイ中にアプリケーションで有効期限の短い分析を実行するために使用されます。**Experiment** CR は個別に使用することも、**Rollout** CR の一部として実行することもできます。
Service と **Ingress**	Argo Rollouts は、Service および Ingress コントローラーを使用して、サービスと Ingress によるトラフィックのルーティングをネイティブにサポートします。
Route および **VirtualService**	OpenShift **Route** および Red Hat OpenShift Service Mesh **VirtualService** リソースは、異なるアプリケーションバージョン間でトラフィック分割を実行するために使用されます。

1.4. ARGO ROLLOUTS CLI の概要

オプションのプラグインである Argo Rollouts CLI を使用すると、OpenShift Container Platform Web コンソールまたは CLI (**oc**) を使用する必要がなくなり、Argo Rollouts リソースを直接管理および監視できます。

Argo Rollouts CLI プラグインを使用すると、次のアクションを実行できます。

- Argo Rollouts イメージに変更を加えます。
- Argo Rollouts プロモーションの進捗を監視します。
- カナリアデプロイメントにおけるプロモーション手順に進みます。
- 失敗した Argo Rollouts デプロイメントを終了します。

Argo Rollouts CLI プラグインは、**oc** および **kubectl** コマンドと直接統合されます。

1.5. 関連情報

- Argo Rollouts のスタートガイド
- RolloutManager カスタムリソースの作成
- Argo Rollouts CLI のインストール
- カナリアデプロイメント
- Blue-Green デプロイメント
- **RolloutManager** カスタムリソースの仕様
- Argo Rollouts による Blue-Green および Canary のデプロイメント
- Argo Rollouts テクノロジープレビューの制限事項

第2章 ARGO ROLLOUTS を使用したデプロイメントのプログレッシブ配信

Argo Rollouts を使用してプログレッシブ配信を管理するには、クラスターに {gitops-titel} Operator をインストールした後、選択した namespace に **RolloutManager** カスタムリソース (CR) インスタンスを作成して設定できます。**RolloutManager** CR のスコープを 1 つまたは複数の namespace に設定できます。

2.1. 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 1.9.0 以降のバージョンがクラスターにインストールされている。

2.2. ROLLOUTMANAGER カスタムリソースの作成

Red Hat OpenShift GitOps で Argo Rollouts を使用してデプロイメントのプログレッシブ配信を管理するには、選択した namespace で **RolloutManager** カスタムリソース (CR) を作成して設定する必要があります。デフォルトでは、新しい **argo-rollouts** インスタンスには、それがデプロイされている namespace でのみリソースを管理する権限がありますが、必要に応じて複数の namespace で Argo Rollouts を使用できます。

前提条件

- Red Hat OpenShift GitOps 1.9.0 以降のバージョンがクラスターにインストールされている。

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. **Administrator** パースペクティブで、**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、**RolloutManager** カスタムリソース (CR) を作成および設定するプロジェクトを作成または選択します。
4. インストールされた Operator から **Red Hat OpenShift GitOps** を選択します。
5. **Details** タブの **Provided APIs** セクションで、**RolloutManager** ペインの **Create instance** をクリックします。
6. **Create RolloutManager** ページで **YAML view** を選択し、デフォルトの YAML を使用するか、要件に応じて編集します。

   例: RolloutManager CR

```
apiVersion: argoproj.io/v1alpha1
kind: RolloutManager
metadata:
  name: argo-rollout
  labels:
    example: basic
spec: {}
```

7. **Create** をクリックします。
8. **RolloutManager** タブの **RolloutManagers** セクションで、RolloutManager インスタンスの **Status** フィールドに **Phase: Available** と表示されていることを確認します。
9. 左側のナビゲーションペインで、namespace をスコープとしたサポートリソースの作成を確認します。
   - **Workloads → Deployments** をクリックして、**argo-rollouts** デプロイメントが使用可能で、**Status** に **1 of 1 pods** が実行中と表示されていることを確認します。
   - **Workloads → Secrets** をクリックして、**argo-rollouts-notification-secret** シークレットが使用可能であることを確認します。
   - **Networking → Services** をクリックして、**argo-rollouts-metrics** サービスが利用可能であることを確認します。
   - **User Management → Roles** をクリックして、**argo-rollouts** ロール、**argo-rollouts-aggregate-to-admin**、**argo-rollouts-aggregate-to-edit**、および **argo-rollouts-aggregate-to-view** クラスターロールが使用可能であることを確認します。
   - **User Management → RoleBindings** をクリックして、**argo-rollouts** ロールバインディングが使用可能であることを確認します。

2.3. ROLLOUTMANAGER カスタムリソースの削除

Red Hat OpenShift GitOps Operator をアンインストールしても、インストール中に作成されたリソースは削除されません。Red Hat OpenShift GitOps Operator をアンインストールする前に、**RolloutManager** カスタムリソース (CR) を手動で削除する必要があります。

前提条件

- Red Hat OpenShift GitOps 1.9.0 以降のバージョンがクラスターにインストールされている。
- **RolloutManager** CR は namespace に存在します。

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. **Administrator** パースペクティブで、**Operators → Installed Operators** をクリックします。
3. **Project** ドロップダウンメニューをクリックし、**RolloutManager** CR を含むプロジェクトを選択します。
4. インストールされた Operator から **Red Hat OpenShift GitOps** を選択します。
5. **RolloutManager** タブをクリックして、**RolloutManagers** セクションで RolloutManager インスタンスを見つけます。
6. インスタンスをクリックします。
7. ドロップダウンメニューから **Actions → Delete RolloutManager** をクリックし、ダイアログボックスで **Delete** をクリックして確認します。

8. **RolloutManager** タブの **RolloutManagers** セクションで、RolloutManager インスタンスが使用できないことを確認します。

9. 左側のナビゲーションペインで、namespace をスコープとするサポートリソースが削除されていることを確認します。

   - **Workloads → Deployments** をクリックして、**argo-rollouts** デプロイメントが削除されていることを確認します。
   - **Workloads → Secrets** をクリックして、**argo-rollouts-notification-secret** シークレットが削除されていることを確認します。
   - **Networking → Services** をクリックして、**argo-rollouts-metrics** サービスが削除されていることを確認します。
   - **User Management → Roles** をクリックして、**argo-rollouts** ロールと、クラスターロール **argo-rollouts-aggregate-to-admin**、**argo-rollouts-aggregate-to-edit**、および **argo-rollouts-aggregate-to-view** が削除されていることを確認します。
   - **User Management → RoleBindings** をクリックして、**argo-rollouts** ロールバインディングが削除されていることを確認します。

2.4. LINUX への ARGO ROLLOUTS CLI のインストール

Linux に Argo Rollouts CLI をインストールできます。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、Argo Rollouts CLI バイナリー **kubectl-argo-rollouts** の最新バージョンをダウンロードします。

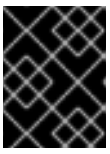
```
$ curl -LO https://github.com/argoproj/argo-rollouts/releases/latest/download/kubectl-argo-rollouts-linux-amd64
```

2. 次のコマンドを実行して、**kubectl-argo-rollouts** バイナリーが実行可能であることを確認します。

```
$ chmod +x ./kubectl-argo-rollouts-linux-amd64
```

3. 次のコマンドを実行して、**kubectl-argo-rollouts** バイナリーをシステムパスに移動します。

```
# mv ./kubectl-argo-rollouts-linux-amd64 /usr/local/bin/kubectl-argo-rollouts
```



重要

このコマンドを実行するには、スーパーユーザー権限が割り当てられていることを確認してください。

4. 次のコマンドを実行して同様の出力を受け取り、プラグインが正しくインストールされていることを確認します。

```
$ oc argo rollouts version
```

出力例

```
kubectl-argo-rollouts: v1.6.6+737ca89
  BuildDate: 2024-02-13T15:39:31Z (1)
  GitCommit: 737ca89b42e4791e96e05b438c2b8540737a2a1a
  GitTreeState: clean
  GoVersion: go1.20.14 (2)
  Compiler: gc
  Platform: linux/amd64 (3)
```

1. Argo Rollouts バイナリーの構築の日付情報。
2. Argo Rollouts バイナリーの構築に使用される Go 言語のバージョン。
3. Argo Rollouts バイナリーの構築に使用されるプラットフォーム。

2.5. MAC OS への ARGO ROLLOUTS CLI のインストール

macOS ユーザーの場合、Homebrew パッケージマネージャーを使用して Argo Rollouts CLI をインストールできます。

前提条件

- Homebrew (**brew**) パッケージマネージャーがインストールされている。

手順

- 次のコマンドを実行して、Argo Rollouts CLI をインストールします。

```
$ brew install argoproj/tap/kubectl-argo-rollouts
```

2.6. 関連情報

- Argo Rollouts のスタートガイド
- Red Hat OpenShift GitOps のインストール
- Red Hat OpenShift GitOps のアンインストール
- **RolloutManager** カスタムリソースの仕様
- Argo Rollouts テクノロジープレビューの制限事項

第3章 ARGO ROLLOUTS のスタートガイド

Argo Rollouts は、カナリア および Blue-Green デプロイメントストラテジー をサポートしています。このガイドでは、カナリアデプロイメントストラテジーを使用した例と手順を示し、ロールアウトのデプロイ、更新、プロモーション、および手動での中止を行うのに役立ちます。

カナリアベースのデプロイメントストラテジーでは、トラフィックを 2 つのアプリケーションバージョン間で分割します。

- **カナリアバージョン**: トラフィックを段階的にルーティングするアプリケーションの新しいバージョン。
- **安定バージョン**: アプリケーションの最新バージョン。カナリアバージョンが安定し、すべてのユーザートラフィックがそのバージョンに送信されるようになると、カナリアバージョンが新しい安定バージョンになります。以前の安定バージョンは破棄されます。

3.1. 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。
- OpenShift Container Platform Web コンソールにアクセスできる。
- OpenShift Container Platform クラスターに Red Hat OpenShift GitOps がインストールされている。
- OpenShift Container Platform クラスターに Argo Rollouts がインストールされている。
- システムに Argo Rollouts CLI がインストールされている。

3.2. ロールアウトのデプロイ

クラスター管理者は、ユーザートラフィックのサブセットを新しいアプリケーションバージョンに段階的にルーティングするように Argo Rollouts を設定できます。次に、アプリケーションがデプロイされ、機能しているかどうかをテストできます。

次の手順例では、**rollouts-demo** ロールアウトとサービスを作成します。次に、ロールアウトではトラフィックの 20% をアプリケーションのカナリアバージョンにルーティングし、手動によるプロモーションを待ちます。次に、トラフィック全体が新しいアプリケーションバージョンにルーティングされるまで、複数の自動プロモーションを実行します。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operator → Installed Operator → Red Hat OpenShift GitOps → Rollout** をクリックします。
2. **Project** ドロップダウンメニューから、**Rollout** カスタムリソース (CR) を作成および設定するプロジェクトを作成または選択します。
3. **Create Rollout** をクリックし、YAML ビューに以下の設定を入力します。

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollouts-demo
spec:
```

```
replicas: 5
strategy:
  canary: (1)
    steps: (2)
    - setWeight: 20 (3)
    - pause: {} (4)
    - setWeight: 40
    - pause: {duration: 45} (5)
    - setWeight: 60
    - pause: {duration: 20}
    - setWeight: 80
    - pause: {duration: 10}
revisionHistoryLimit: 2
selector:
  matchLabels:
    app: rollouts-demo
template: (6)
  metadata:
    labels:
      app: rollouts-demo
  spec:
    containers:
    - name: rollouts-demo
      image: argoproj/rollouts-demo:blue
      ports:
      - name: http
        containerPort: 8080
        protocol: TCP
      resources:
        requests:
          memory: 32Mi
          cpu: 5m
```

(1) ロールアウトで使用する必要のあるデプロイメントストラテジー。

(2) ロールアウトの手順を指定します。この例では、トラフィックの 20%、40%、60%、80% を段階的にカナリアバージョンにルーティングします。

(3) カナリアバージョンに送信する必要があるトラフィックの割合。値が 20 の場合、トラフィックの 20% がカナリアバージョンに送信されることを意味します。

(4) プロモーションのリクエストが見つかるまで Argo Rollouts コントローラーに、無期限に一時停止するように指定します。

(5) Argo Rollouts コントローラーに 45 秒間一時停止するように指定します。期間の値は、秒 (**s**)、分 (**m**)、または時間 (**h**) で設定できます。たとえば、1 時間の場合は **1h** と指定できます。値が指定されていない場合、期間の値はデフォルトで秒になります。

(6) 作成する Pod を指定します。

4. **Create** をクリックします。

注記

ロールアウトが作成時にすぐに利用できるように、Argo Rollouts コントローラーは、**.spec.template.spec.containers.image** フィールドで指定された **argoproj/rollouts-demo:blue** の初期コンテナーイメージを安定バージョンとして自動的に扱います。最初のインスタンスでは、**Rollout** リソースの作成により、すべてのトラフィックがアプリケーションの安定バージョンにルーティングされ、トラフィックがカナリアバージョンに送信される部分がスキップされます。ただし、**.spec.template.spec.containers.image** フィールドに変更を加えた以降のすべてのアプリケーションアップグレードでは、Argo Rollouts コントローラーは通常どおりカナリアの手順を実行します。

5. 次のコマンドを実行して、ロールアウトが正しく作成されたことを確認します。

```
$ oc argo rollouts list rollouts -n <namespace> 1
```

1 **Rollout** リソースが定義されている namespace を指定します。

出力例

```
NAME          STRATEGY   STATUS     STEP  SET-WEIGHT  READY  DESIRED  UP-TO-DATE  AVAILABLE
rollouts-demo  Canary    Healthy    8/8   100         5/5    5        5           5
```

6. **rollouts-demo** ロールアウトをターゲットとする Kubernetes サービスを作成します。

   a. Web コンソールの **Administrator** パースペクティブで、**Networking → Services** をクリックします。

   b. **Create Service** をクリックし、YAML ビューに次の設定を入力します。

```
apiVersion: v1
kind: Service
metadata:
  name: rollouts-demo
spec:
  ports: 1
  - port: 80
    targetPort: http
    protocol: TCP
    name: http

  selector: 2
    app: rollouts-demo
```

   1 コンテナー内で実行するためにアプリケーションが使用するポートの名前を指定します。

   2 **selector** フィールドの内容が **Rollout** カスタムリソース (CR) の内容と同じであることを確認します。

   c. **Create** をクリックします。

ロールアウトは、作成されたサービスをカナリア **ReplicaSet** の Pod テンプレートハッシュで自動的に更新します。例: **rollouts-pod-template-hash: 687d76d795**

7. 次のコマンドを実行して、ロールアウトの進行状況を確認します。

```
$ oc argo rollouts get rollout rollouts-demo --watch -n <namespace>
```
(1)

(1) **Rollout** リソースが定義されている namespace を指定します。

出力例

```
Name:            rollouts-demo
Namespace:       spring-petclinic
Status:          ✔ Healthy
Strategy:        Canary
  Step:          8/8
  SetWeight:     100
  ActualWeight:  100
Images:          argoproj/rollouts-demo:blue (stable)
Replicas:
  Desired:       5
  Current:       5
  Updated:       5
  Ready:         5
  Available:     5

NAME                                       KIND        STATUS     AGE    INFO
⟳ rollouts-demo                            Rollout     ✔ Healthy  4m50s
└──# revision:1
   └──  rollouts-demo-687d76d795           ReplicaSet  ✔ Healthy  4m50s  stable
      ├──□ rollouts-demo-687d76d795-75k57  Pod         ✔ Running  4m49s  ready:1/1
      ├──□ rollouts-demo-687d76d795-bv5zf  Pod         ✔ Running  4m49s  ready:1/1
      ├──□ rollouts-demo-687d76d795-jsxg8  Pod         ✔ Running  4m49s  ready:1/1
      ├──□ rollouts-demo-687d76d795-rsgtv  Pod         ✔ Running  4m49s  ready:1/1
      └──□ rollouts-demo-687d76d795-xrmrj  Pod         ✔ Running  4m49s  ready:1/1
```

ロールアウトの作成後、ロールアウトの **Status** フィールドに **Phase: Healthy** が表示されることを確認できます。

8. **Rollout** タブの **Rollouts** セクションで、**rollouts-demo** ロールアウトの **Status** フィールドに **Phase: Healthy** が表示されていることを確認します。

ヒント

または、以下のコマンドを実行して、ロールアウトが正常であることを確認できます。

```
$ oc argo rollouts status rollouts-demo -n <namespace>
```
(1)

(1) **Rollout** リソースが定義されている namespace を指定します。

出力例

```
Healthy
```

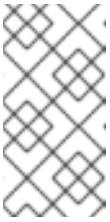
これで、**Rollout** CR の次の更新でカナリアデプロイメントを実行する準備が整いました。

3.3. ロールアウトの更新

.spec.template.spec フィールド (コンテナーイメージバージョンなど) を変更して **Rollout** カスタムリソース (CR) を更新すると、更新されたコンテナーイメージバージョンを使用して、**ReplicaSet** で新しい Pod が作成されます。

手順

1. ロールアウトでデプロイされたコンテナーイメージを変更して、アプリケーションの新しいカナリアバージョンをシミュレートします。
   a. Web コンソールの **Administrator** パースペクティブで、**Operator → Installed Operator → Red Hat OpenShift GitOps → Rollout** に移動します。
   b. 既存の **rollouts-demo** ロールアウトを選択し、YAML ビューで **.spec.template.spec.containers.image** の値を **argoproj/rollouts-demo:blue** から **argoproj/rollouts-demo:yellow** に変更します。
   c. **Save**、**Reload** の順にクリックします。
   ロールアウトでデプロイされたコンテナーイメージが変更され、ロールアウトによって新しいカナリアデプロイメントが開始されます。



注記

Rollout CR の **.spec.strategy.canary.steps** フィールドで定義されている **setWeight** プロパティーに従って、最初にルートへのトラフィックの 20% がカナリアバージョンに到達し、ロールアウトは、プロモーションのリクエストが受信されるまで無期限に一時停止されます。

トラフィックの 20% がカナリアバージョンに向けられ、後続のステップでプロモーションのリクエストが指定されるまでロールアウトが無期限に一時停止されるルートの例

```
spec:
  replicas: 5
  strategy:
    canary: (1)
      steps: (2)
      - setWeight: 20 (3)
      - pause: {} (4)
  # (...)
```

(1) ロールアウトで使用する必要のあるデプロイメントストラテジー。

(2) ロールアウトの手順です。この例では、トラフィックの 20%、40%、60%、80% を段階的にカナリアバージョンにルーティングします。

(3) カナリアバージョンに送信する必要があるトラフィックの割合。値が 20 の場合、トラフィックの 20% がカナリアバージョンに送信されることを意味します。

(4) プロモーションのリクエストが見つかるまで、Argo Rollouts コントローラーを無期限に一時停止する仕様。

2. 次のコマンドを実行して、ロールアウトの進行状況を確認します。

```
$ oc argo rollouts get rollout rollouts-demo --watch -n <namespace> 1
```

1 **Rollout** CR が定義されている namespace を指定します。

出力例

```
Name:            rollouts-demo
Namespace:       spring-petclinic
Status:          Paused
Message:         CanaryPauseStep
Strategy:        Canary
  Step:          1/8
  SetWeight:     20
  ActualWeight:  20
Images:          argoproj/rollouts-demo:blue (stable)
                 argoproj/rollouts-demo:yellow (canary)
Replicas:
  Desired:       5
  Current:       5
  Updated:       1
  Ready:         5
  Available:     5

NAME                                  KIND        STATUS     AGE    INFO
⟳ rollouts-demo                       Rollout     Paused     9m51s
├──# revision:2
│  └──  rollouts-demo-6cf78c66c5      ReplicaSet  ✔ Healthy  99s    canary
│     └──□ rollouts-demo-6cf78c66c5-zrgd4  Pod    ✔ Running  98s    ready:1/1
└──# revision:1
   └──  rollouts-demo-687d76d795      ReplicaSet  ✔ Healthy  9m51s  stable
      ├──□ rollouts-demo-687d76d795-75k57  Pod    ✔ Running  9m50s  ready:1/1
      ├──□ rollouts-demo-687d76d795-jsxg8  Pod    ✔ Running  9m50s  ready:1/1
      ├──□ rollouts-demo-687d76d795-rsgtv  Pod    ✔ Running  9m50s  ready:1/1
      └──□ rollouts-demo-687d76d795-xrmrj  Pod    ✔ Running  9m50s  ready:1/1
```

ロールアウトの更新ストラテジー設定で一時停止期間が指定されていないため、ロールアウトは現在一時停止状態になっています。

3. 前の手順を繰り返して、新しくデプロイされたアプリケーションのバージョンをテストし、期待どおりに動作することを確認します。たとえば、ブラウザーを使用してアプリケーションの操作、アプリケーションの検証、テストの実行、コンテナーログの確認を行います。
ロールアウトは、次のステップに進むまで、一時停止されたままになります。

新しいバージョンのアプリケーションが期待どおりに動作していることを確認したら、プロモーションを続行するか、ロールアウトを中止するかを決定できます。したがって、「ロールアウトのプロモート」または「ロールアウトの手動中断」の手順に従ってください。

3.4. ロールアウトのプロモート

ロールアウトは現在一時停止状態にあるため、クラスター管理者は、ロールアウトを手動でプロモートさせて次のステップに進むことができるようにする必要があります。

手順

1. Argo Rollouts CLI で次のコマンドを実行して、アプリケーションの別のカナリアバージョンを新たにシミュレートします。

```
$ oc argo rollouts promote rollouts-demo -n <namespace> 1
```

1 **Rollout** リソースが定義されている namespace を指定します。

出力例

```
rollout 'rollouts-demo' promoted
```

これにより、カナリアバージョンではトラフィックの重みが 40% に増加します。

2. 次のコマンドを実行して、ロールアウトが残りの手順で進行していることを確認します。

```
$ oc argo rollouts get rollout rollouts-demo -n <namespace> --watch 1
```

1 **Rollout** リソースが定義されている namespace を指定します。

Rollout CR で定義されている残りのステップには、**pause: {duration: 45}** などのように期間が設定されているため、Argo Rollouts コントローラーは指定の期間待機してから、自動的に次のステップに進みます。

すべての手順が正常に完了すると、新しい **ReplicaSet** オブジェクトが安定したレプリカセットとしてマークされます。

出力例

```
Name:            rollouts-demo
Namespace:       spring-petclinic
Status:          ✔ Healthy
Strategy:        Canary
  Step:          8/8
  SetWeight:     100
  ActualWeight:  100
Images:          argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       5
  Current:       5
  Updated:       5
  Ready:         5
  Available:     5

NAME                                KIND        STATUS        AGE   INFO
⟳ rollouts-demo                     Rollout     ✔ Healthy     14m
├──# revision:2
│  └──  rollouts-demo-6cf78c66c5          ReplicaSet  ✔ Healthy     6m5s  stable
│     ├──□ rollouts-demo-6cf78c66c5-zrgd4  Pod         ✔ Running     6m4s  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-g9kd5  Pod         ✔ Running     2m4s  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-2ptpp  Pod         ✔ Running     78s   ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-tmk6c  Pod         ✔ Running     58s   ready:1/1
```

```
│    └──□ rollouts-demo-6cf78c66c5-zv6lx  Pod         ✔ Running     47s   ready:1/1
└──# revision:1
   └──  rollouts-demo-687d76d795          ReplicaSet  • ScaledDown  14m
```

3.5. ロールアウトの手動中断

カナリアデプロイメントを使用する場合、ロールアウトではアプリケーションの初期カナリアバージョンがデプロイされます。手動またはプログラム的に検証できます。カナリアバージョンを検証して安定バージョンにプロモートすると、新しい安定バージョンがすべてのユーザーに公開されます。

ただし、カナリアバージョンでバグ、エラー、またはデプロイメントの問題が発見される場合があり、カナリアロールアウトを中止して、アプリケーションの安定したバージョンにロールバックする必要がある場合があります。

カナリアロールアウトを中止すると、新しいカナリアバージョンのリソースが削除され、アプリケーションの以前の安定バージョンが復元されます。カナリアに送信されていた Ingress、ルート、仮想サービスなどのすべてのネットワークトラフィックは、元の安定バージョンに戻ります。

次のサンプル手順では、アプリケーションの新しい **red** カナリアバージョンをデプロイし、完全に安定バージョンにプロモートする前に中止します。

手順

1. Argo Rollouts CLI で次のコマンドを実行して、コンテナーイメージのバージョンを更新し、**.spec.template.spec.containers.image** の値を **argoproj/rollouts-demo:yellow** から **argoproj/rollouts-demo:red** に変更します。

```
$ oc argo rollouts set image rollouts-demo rollouts-demo=argoproj/rollouts-demo:red -n
<namespace> 1
```

1 **ロールアウト** カスタムリソース (CR) が定義されている namespace を指定します。

出力例

```
rollout "rollouts-demo" image updated
```

ロールアウトでデプロイされたコンテナーイメージが変更され、ロールアウトによって新しいカナリアデプロイメントが開始されます。

2. ロールアウトが一時停止状態になるまで待ちます。

3. 次のコマンドを実行して、ロールアウトによって **rollouts-demo:red** カナリアバージョンがデプロイされ、一時停止ステータスに到達したことを確認します。

```
$ oc argo rollouts get rollout rollouts-demo --watch -n <namespace> 1
```

1 **Rollout** CR が定義されている namespace を指定します。

出力例

```
Name:            rollouts-demo
Namespace:       spring-petclinic
```

```
Status:          Paused
Message:         CanaryPauseStep
Strategy:        Canary
  Step:          1/8
  SetWeight:     20
  ActualWeight:  20
Images:          argoproj/rollouts-demo:red (canary)
                 argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       5
  Current:       5
  Updated:       1
  Ready:         5
  Available:     5

NAME                                  KIND        STATUS        AGE    INFO
⟳ rollouts-demo                       Rollout     Paused        17m
├──# revision:3
│  └──── rollouts-demo-5747959bdb     ReplicaSet  ✔ Healthy     75s    canary
│     └──□ rollouts-demo-5747959bdb-fdrsg  Pod     ✔ Running     75s    ready:1/1
├──# revision:2
│  └──── rollouts-demo-6cf78c66c5     ReplicaSet  ✔ Healthy     9m45s  stable
│     ├──□ rollouts-demo-6cf78c66c5-zrgd4  Pod     ✔ Running     9m44s  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-2ptpp  Pod     ✔ Running     4m58s  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-tmk6c  Pod     ✔ Running     4m38s  ready:1/1
│     └──□ rollouts-demo-6cf78c66c5-zv6lx  Pod     ✔ Running     4m27s  ready:1/1
└──# revision:1
   └──── rollouts-demo-687d76d795     ReplicaSet  • ScaledDown  17m
```

4. 以下のコマンドを実行してロールアウトの更新を停止します。

```
$ oc argo rollouts abort rollouts-demo -n <namespace> 1
```

1 **Rollout** CR が定義されている namespace を指定します。

出力例

```
rollout 'rollouts-demo' aborted
```

Argo Rollouts コントローラーは、アプリケーションのカナリアリソースを削除し、安定バージョンにロールバックします。

5. ロールアウトを中止した後、次のコマンドを実行して、カナリア **ReplicaSet** が 0 レプリカにスケーリングされていることを確認します。

```
$ oc argo rollouts get rollout rollouts-demo --watch -n <namespace> 1
```

1 **Rollout** CR が定義されている namespace を指定します。

出力例

```
Name:          rollouts-demo
```

```
Namespace:       spring-petclinic
Status:          ✖ Degraded
Message:         RolloutAborted: Rollout aborted update to revision 3
Strategy:        Canary
  Step:          0/8
  SetWeight:     0
  ActualWeight:  0
Images:          argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       5
  Current:       5
  Updated:       0
  Ready:         5
  Available:     5

NAME                                   KIND        STATUS        AGE    INFO
⟳ rollouts-demo                        Rollout     ✖ Degraded    24m
├──# revision:3
│  └──  rollouts-demo-5747959bdb       ReplicaSet  • ScaledDown  7m38s  canary
├──# revision:2
│  └──  rollouts-demo-6cf78c66c5       ReplicaSet  ✔ Healthy     16m    stable
│     ├──□ rollouts-demo-6cf78c66c5-zrgd4  Pod      ✔ Running     16m    ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-2ptpp  Pod      ✔ Running     11m    ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-tmk6c  Pod      ✔ Running     11m    ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-zv6lx  Pod      ✔ Running     10m    ready:1/1
│     └──□ rollouts-demo-6cf78c66c5-mlbsh  Pod      ✔ Running     4m47s  ready:1/1
└──# revision:1
   └──  rollouts-demo-687d76d795       ReplicaSet  • ScaledDown  24m
```

ロールアウトステータスは **Degraded** としてマークされています。これは、アプリケーションが以前の安定バージョン (**yellow)** にロールバックされているにもかかわらず、ロールアウトが現在、**.spec.template.spec.containers.image** フィールド内で設定された、希望のバージョン (**red**) ではないことを示しています。

注記

Degraded ステータスは、アプリケーションの正常性を反映しません。これは、必要なコンテナーイメージのバージョンと実行中のコンテナーイメージのバージョンが一致していないことを示しているだけです。

6. コンテナーイメージのバージョンを以前の安定バージョン **yellow** に更新し、次のコマンドを実行して **.spec.template.spec.containers.image** の値を変更します。

```
$ oc argo rollouts set image rollouts-demo rollouts-demo=argoproj/rollouts-demo:yellow -n
<namespace> 1
```

1 **Rollout** CR が定義されている namespace を指定します。

出力例

```
rollout "rollouts-demo" image updated
```

ロールアウトでは、分析とプロモーションの手順がスキップされ、以前の安定したバージョン **yellow** にロールバックされ、安定した **ReplicaSet** のデプロイを早い段階で行うことができます。

7. 次のコマンドを実行して、ロールアウトのステータスが **Healthy** としてマークされていることを確認します。

```
$ oc argo rollouts get rollout rollouts-demo --watch -n <namespace>
```
(1)

(1) **Rollout** CR が定義されている namespace を指定します。

出力例

```
Name:            rollouts-demo
Namespace:       spring-petclinic
Status:          ✔ Healthy
Strategy:        Canary
  Step:          8/8
  SetWeight:     100
  ActualWeight:  100
Images:          argoproj/rollouts-demo:yellow (stable)
Replicas:
  Desired:       5
  Current:       5
  Updated:       5
  Ready:         5
  Available:     5

NAME                                  KIND        STATUS        AGE  INFO
⟳ rollouts-demo                       Rollout     ✔ Healthy     63m
├──# revision:4
│  └──  rollouts-demo-6cf78c66c5          ReplicaSet  ✔ Healthy     55m  stable
│     ├──□ rollouts-demo-6cf78c66c5-zrgd4  Pod         ✔ Running     55m  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-2ptpp  Pod         ✔ Running     50m  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-tmk6c  Pod         ✔ Running     50m  ready:1/1
│     ├──□ rollouts-demo-6cf78c66c5-zv6lx  Pod         ✔ Running     50m  ready:1/1
│     └──□ rollouts-demo-6cf78c66c5-mlbsh  Pod         ✔ Running     44m  ready:1/1
├──# revision:3
│  └──  rollouts-demo-5747959bdb          ReplicaSet  • ScaledDown  46m
└──# revision:1
   └──  rollouts-demo-687d76d795          ReplicaSet  • ScaledDown  63m
```

3.6. 関連情報

- Red Hat OpenShift GitOps のインストール
- Red Hat OpenShift GitOps のアンインストール
- **RolloutManager** カスタムリソースの仕様
- Argo Rollouts CLI の概要
- Argo Rollouts テクノロジープレビューの制限事項

第4章 ARGO ROLLOUTS を使用したトラフィックのルーティング

Argo Rollouts とそのトラフィック分割メカニズムを使用すると、ユーザートラフィックのサブセットを新しいアプリケーションバージョンに段階的にルーティングできます。次に、アプリケーションがデプロイされ、機能しているかどうかをテストできます。

Openshift Routes を使用すると、要件に基づいてクラスター環境内のさまざまなアプリケーションにトラフィックを送信することで、トラフィックの量を減らしたり増やしたりするように Argo Rollouts を設定できます。

OpenShift Routes を使用して、2 つのアプリケーションバージョン間でトラフィックを分割できます。

- **カナリアバージョン**: トラフィックを段階的にルーティングするアプリケーションの新しいバージョン。
- **安定バージョン**: アプリケーションの最新バージョン。カナリアバージョンが安定し、すべてのユーザートラフィックがそのバージョンに送信されるようになると、カナリアバージョンが新しい安定バージョンになります。以前の安定バージョンは破棄されます。

4.1. 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。
- OpenShift Container Platform クラスターに Red Hat OpenShift GitOps がインストールされている。
- OpenShift Container Platform クラスターに Argo Rollouts がインストールされている。
- システムに Red Hat OpenShift GitOps CLI がインストールされている。
- システムに Argo Rollouts CLI がインストールされている。

4.2. OPENSHIFT ROUTES を使用してトラフィックをルーティングするように ARGO ROLLOUTS を設定する

OpenShift Routes を使用して、ルート、ロールアウト、およびサービスを作成するように Argo Rollouts を設定できます。

以下の手順例では、ルート、ロールアウト、および 2 つのサービスを作成します。その後、徐々にトラフィックの割合を増やしてアプリケーションのカナリアバージョンにルーティングし、そのカナリア状態が成功としてマークされて新しい安定バージョンになります。

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。
- OpenShift Container Platform クラスターに Red Hat OpenShift GitOps がインストールされている。
- OpenShift Container Platform クラスターに Argo Rollouts がインストールされている。詳細は、「RolloutManager カスタムリソースの作成」を参照してください。
- システムに Red Hat OpenShift GitOps CLI がインストールされている。詳細は、「GitOps CLI のインストール」を参照してください。

- システムに Argo Rollouts CLI がインストールされている。詳細は、「Argo Rollouts CLI の概要」を参照してください。

手順

1. **Route** オブジェクトを作成します。
   a. Web コンソールの **Administrator** パースペクティブで、**Networking → Routes** をクリックします。
   b. **Create Route** をクリックします。
   c. **Create Route** ページで、**YAML view** をクリックし、以下のスニペットを追加します。以下の例では、**rollouts-demo-route** というルートを作成します。

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: rollouts-demo-route
spec:
  port:
    targetPort: http (1)
  tls: (2)
    insecureEdgeTerminationPolicy: Redirect
    termination: edge
  to:
    kind: Service
    name: argo-rollouts-stable-service (3)
    weight: 100 (4)

  alternateBackends:
    - kind: Service
      name: argo-rollouts-canary-service (5)
      weight: 0 (6)
```

(1) コンテナー内で実行するためにアプリケーションが使用するポートの名前を指定します。

(2) ルートを保護するために使用される TLS 設定を指定します。

(3) 対象となる安定したサービスの名前。

(4) このフィールドは、Route Rollout プラグインによって安定した重みに自動的に変更されます。

(5) 対象となるカナリアサービスの名前。

(6) このフィールドは、Route Rollout プラグインにより、カナリアの重みに自動的に変更されます。

   d. **Create** をクリックしてルートを作成します。その後、**Routes** ページに表示されます。

2. ルートで参照されるサービス (カナリアおよび安定) を作成します。
   a. Web コンソールの **Administrator** パースペクティブで、**Networking → Services** をクリッ

クします。

b. **Create Service** をクリックします。

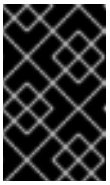
c. **Create Service** ページで、**YAML view** をクリックし、以下のスニペットを追加します。以下の例では、**argo-rollouts-canary-service** という名前のカナリアサービスを作成します。カナリアトラフィックはこのサービスに転送されます。

```
apiVersion: v1
kind: Service
metadata:
  name: argo-rollouts-canary-service
spec:
  ports: 1
  - port: 80
    targetPort: http
    protocol: TCP
    name: http

  selector: 2
    app: rollouts-demo
```

1 コンテナー内で実行するためにアプリケーションが使用するポートの名前を指定します。

2 **selector** フィールドの内容が、安定したサービスおよび **Rollout** カスタムリソース (CR) と同じであることを確認します。



重要

Route オブジェクトで指定されたカナリアサービスの名前が、**Service** オブジェクトで指定されたカナリアサービスの名前と一致していることを確認します。

d. **Create** をクリックしてカナリアサービスを作成します。
ロールアウトは、作成されたサービスをカナリア **ReplicaSet** の Pod テンプレートハッシュで自動的に更新します。例: **rollouts-pod-template-hash: 7bf84f9696**。

e. これらの手順を繰り返して安定したサービスを作成します。次の例では **argo-rollouts-stable-service** という安定したサービスを作成します。安定したトラフィックがこのサービスに送られます。

```
apiVersion: v1
kind: Service
metadata:
  name: argo-rollouts-stable-service
spec:
  ports: 1
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
```

```
    selector: 2
      app: rollouts-demo
```

1 コンテナー内で実行するためにアプリケーションが使用するポートの名前を指定します。

2 **selector** フィールドの内容がカナリアサービスと **Rollout** CR と同じであることを確認します。

重要

Route オブジェクトで指定された安定したサービスの名前が、**Service** オブジェクトで指定された安定したサービスの名前と一致していることを確認します。

f. **Create** をクリックして安定したサービスを作成します。
ロールアウトは、安定した **ReplicaSet** の Pod テンプレートハッシュを使用して、作成されたサービスを自動的に更新します。例: **rollouts-pod-template-hash: 1b6a7733**。

3. **Route** および **Service** オブジェクトを参照する **Rollout** CR を作成します。

a. Web コンソールの **Administrator** パースペクティブで、**Operator → Installed Operator → Red Hat OpenShift GitOps → Rollout** に移動します。

b. **Create Rollout** ページで、**YAML view** をクリックし、次のスニペットを追加します。次の例では、**rollouts-demo** という **Rollout** CR を作成します。

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollouts-demo
spec:
  template: 1
    metadata:
      labels:
        app: rollouts-demo
    spec:
      containers:
      - name: rollouts-demo
        image: argoproj/rollouts-demo:blue
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
        resources:
          requests:
            memory: 32Mi
            cpu: 5m

  revisionHistoryLimit: 2
  replicas: 5
  strategy:
    canary:
```

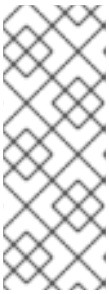
```
    canaryService: argo-rollouts-canary-service 2
    stableService: argo-rollouts-stable-service 3
    trafficRouting:
      plugins:
        argoproj-labs/openshift:
          routes:
            - rollouts-demo-route 4
    steps: 5
    - setWeight: 30
    - pause: {}
    - setWeight: 60
    - pause: {}
  selector: 6
    matchLabels:
      app: rollouts-demo
```

1. 作成する Pod を指定します。
2. この値は、作成されたカナリア **Service** の名前に一致する必要があります。
3. この値は、作成された安定した **Service** の名前と一致する必要があります。
4. この値は、作成された **Route** CR の名前に一致する必要があります。
5. ロールアウトの手順を指定します。この例では、トラフィックの 30%、60%、100% を段階的にカナリアバージョンにルーティングします。
6. **selector** フィールドの内容が、カナリアサービスおよび安定したサービスと同じであることを確認します。

c. **Create** をクリックします。

d. **Rollout** タブの **Rollout** セクションで、ロールアウトの **Status** フィールドに **Phase: Healthy** と表示されていることを確認します。

4. ルートがトラフィックの 100% をアプリケーションの安定バージョンに送信していることを確認します。



注記

Rollout リソースの最初のインスタンスが作成されると、ロールアウトによって、安定バージョンおよびカナリアバージョンのアプリケーションに送信されるトラフィックの量が調整されます。最初のインスタンスでは、**Rollout** リソースの作成により、すべてのトラフィックがアプリケーションの安定バージョンにルーティングされ、トラフィックがカナリアバージョンに送信される部分がスキップされます。

a. **Networking → Routes** に移動し、検証する **Route** リソースを探します。

b. **YAML** タブを選択し、以下のスニペットを表示します。

例: Route

```
kind: Route
```

```
metadata:
  name: rollouts-demo-route
spec:
  alternateBackends:
  - kind: Service
    name: argo-rollouts-canary-service
    weight: 0 (1)
  # (...)
  to:
    kind: Service
    name: argo-rollouts-stable-service
    weight: 100 (2)
```

(1) 値が **0** の場合は、トラフィックの 0% がカナリアバージョンに転送されることを意味します。

(2) 値が **100** の場合、トラフィックの 100% が安定バージョンに送信されることを意味します。

5. ロールアウトでデプロイされたコンテナーイメージを変更して、アプリケーションの新しいカナリアバージョンをシミュレートします。

   a. Web コンソールの **Administrator** パースペクティブで、**Operator → Installed Operator → Red Hat OpenShift GitOps → Rollout** に移動します。

   b. 既存の **Rollout** を選択し、**.spec.template.spec.containers.image** の値を **argoproj/rollouts-demo:blue** から **argoproj/rollouts-demo:yellow** に変更します。
   その結果、ロールアウトにデプロイされたコンテナーイメージが変更され、ロールアウトによって新規のカナリアデプロイメントが開始されます。

注記

Rollout リソースの **.spec.strategy.canary.steps** フィールドで定義されている **setWeight** プロパティーに従って、最初はルートへのトラフィックの 30% がカナリアバージョンに到達し、トラフィックの 70% が安定バージョンに向けられます。トラフィックの 30% がカナリアバージョンに転送されると、ロールアウトは一時停止されます。

トラフィックの 30% がカナリアバージョンに送られ、70% が安定バージョンに送られるルートの例。

```
spec:
  alternateBackends:
  - kind: Service
    name: argo-rollouts-canary-service
    weight: 30
  # (...)
  to:
    kind: Service
    name: argo-rollouts-stable-service
    weight: 70
```

6. Argo Rollouts CLI で次のコマンドを実行して、アプリケーションの別のカナリアバージョンを新たにシミュレートします。

```
$ oc argo rollouts promote rollouts-demo -n <namespace>
```
(1)

(1) **Rollout** リソースが定義されている namespace を指定します。

これにより、トラフィックの重みは、カナリアバージョンでは 60%、安定バージョンでは 40% に増加します。

トラフィックの 60% がカナリアバージョンに送られ、40% が安定バージョンに送られるルートの例。

```
spec:
  alternateBackends:
  - kind: Service
    name: argo-rollouts-canary-service
    weight: 60
  # (...)
  to:
    kind: Service
    name: argo-rollouts-stable-service
    weight: 40
```

7. 次のコマンドを実行して、カナリアバージョンのトラフィックの重みを 100% に増やし、アプリケーションの古い安定バージョンのトラフィックを破棄します。

```
$ oc argo rollouts promote rollouts-demo -n <namespace>
```
(1)

(1) **Rollout** リソースが定義されている namespace を指定します。

トラフィックの 0% がカナリアバージョンに送られ、100% が安定バージョンに送られるルートの例。

```
spec:
  # (...)
  to:
    kind: Service
    name: argo-rollouts-stable-service
    weight: 100
```

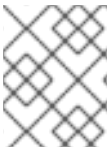
4.3. 関連情報

- GitOps CLI の設定
- Argo Rollouts CLI の概要
- Argo Rollouts テクノロジープレビューの制限事項

第5章 NAMESPACE スコープの ARGO ROLLOUTS インストールサポートの有効化

Red Hat OpenShift GitOps では、Argo Rollouts インストールの 2 つのモードがサポートされます。

- **クラススコープのインストール** (デフォルト): 任意の namespace で定義された Argo Rollouts カスタムリソース (CR) は、Argo Rollouts インスタンスによって調整されます。その結果、クラスター上の任意の namespace で Argo Rollouts CR を使用できます。
- **namespace スコープのインストール**: Argo Rollouts インスタンスは特定の namespace にインストールされ、同じ namespace 内の Argo Rollouts CR のみを処理します。このインストールモードには、以下の利点があります。
  - このモードでは、クラスター全体の **ClusterRole** または **ClusterRoleBinding** パーミッションは必要ありません。クラスター権限を必要とせずに、単一の namespace 内で Argo Rollouts をインストールして使用できます。
  - このモードでは、単一の Argo Rollouts インスタンスのクラススコープを特定の namespace に制限することで、セキュリティー上の利点があります。



注記

意図しない権限昇格を防ぐため、Red Hat OpenShift GitOps では、Argo Rollout のインストールは、一度に 1 つのモードでのみ利用できます。

クラススコープと namespace スコープの Argo Rollouts インストールを切り替えるには、次の手順を実行します。

5.1. NAMESPACE スコープの ARGO ROLLOUTS インストールの設定

Argo Rollouts インストールの namespace スコープのインスタンスを設定するには、次の手順を実行します。

前提条件

- Red Hat OpenShift GitOps クラスターに管理者としてログインしている。
- Red Hat OpenShift GitOps クラスターに Red Hat OpenShift GitOps がインストールされている。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Administration → CustomResourceDefinitions** に移動します。
2. **Subscription** を検索し、**Subscription** CRD をクリックします。
3. **Instances** タブをクリックし、**openshift-gitops-operator** サブスクリプションをクリックします。
4. **YAML** タブをクリックし、YAML ファイルを編集します。
   a. **.spec.config.env** プロパティーで値を **true** に設定して、**NAMESPACE_SCOPED_ARGO_ROLLOUTS** 環境変数を指定します。

namespace スコープの Argo Rollouts のインストール設定の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
spec:
  # (...)
  config:
    env:
      - name: NAMESPACE_SCOPED_ARGO_ROLLOUTS
        value: 'true' (1)
```

(1) 値を **'true'** に設定すると、namespace スコープのインストールが有効になります。値が **'false'** に設定されているか指定されていない場合、インストールはデフォルトでクラスタースコープモードになります。

b. **Save** をクリックします。
Red Hat OpenShift GitOps Operator は、namespace スコープのインストール内での Argo Rollouts カスタムリソースの調整を容易にします。

5. GitOps コンテナーのログを表示して、Red Hat OpenShift GitOps Operator が namespace スコープの Argo Rollouts インストールを有効にしたことを確認します。

a. Web コンソールの **Administrator** パースペクティブで、**Workloads → Pods** に移動します。

b. **openshift-gitops-operator-controller-manager** Pod をクリックし、**Logs** タブをクリックします。

c. **Running in namespaced-scoped mode** のログステートメントを探します。このステートメントは、Red Hat OpenShift GitOps Operator が namespace スコープの Argo Rollouts インストールを有効にしたことを示します。

6. **RolloutManager** リソースを作成して、namespace スコープの Argo Rollouts のインストールを完了します。

a. **Operator → Installed Operator → Red Hat OpenShift GitOps** に移動し、**RolloutManager** タブをクリックします。

b. **Create RolloutManager** を作成します。

c. **YAML view** を選択し、以下のスニペットを入力します。

namespace スコープの Argo Rollouts インストール用の RolloutManager CR の例

```
apiVersion: argoproj.io/v1alpha1
kind: RolloutManager
metadata:
  name: rollout-manager
  namespace: my-application (1)
spec:
  namespaceScoped: true
```

1 namespace スコープの Argo Rollouts インスタンスをインストールするプロジェクトの名前を指定します。

d. **Create** をクリックします。
RolloutManager CR が作成されると、Red Hat OpenShift GitOps は namespace スコープの Argo Rollouts インスタンスを選択した名前空間にインストールします。

7. namespace スコープのインストールが成功していることを確認します。

a. **RolloutManager** タブの **RolloutManagers** セクションで、**RolloutManager** インスタンスの **Status** フィールドが **Phase: Available** であることを確認します。

b. **RolloutManagers** セクションの **YAML** タブで以下の出力を確認して、インストールが正常に行われていることを確認します。

namespace スコープの Argo Rollouts インストール YAML ファイルの例

```
spec:
  namespaceScoped: true
status:
  conditions:
    lastTransitionTime: '2024-07-10T14:20:5z`
    message: ''
    reason: Success
    status: 'True' (1)
    type: 'Reconciled'
  phase: Available
  rolloutController: Available
```

1 このステータスは、namespace スコープの Argo Rollouts インストールが正常に有効化されていることを示します。

クラスター上にクラスタースコープのインストールがすでに存在する場合に、namespace 固有の Argo Rollouts インスタンスをインストールしようとすると、次のエラーメッセージが表示されます。

エラーメッセージが表示される誤ったインストールの例

```
spec:
  namespaceScoped: true
status:
  conditions:
    lastTransitionTime: '2024-07-10T14:10:7z`
    message: 'when Subscription has environment variable
NAMESPACE_SCOPED_ARGO_ROLLOUTS set to False, there may not exist any
namespace-scoped RolloutManagers: only a single cluster-scoped RolloutManager is
supported'
    reason: InvalidRolloutManagerScope
    status: 'False' (1)
    type: 'Reconciled'
  phase: Failure
  rolloutController: Failure
```

1 このステータスは、namespace スコープの Argo Rollouts のインストールが正常に有効になっていないことを示します。インストールのデフォルトは cluster-scoped モー