OpenShift Container Platform 4.16

エッジコンピューティング

ネットワークエッジで OpenShift Container Platform クラスターを設定およびデプロイする

Last Updated: 2024-07-03

OpenShift Container Platform 4.16 エッジコンピューティング

ネットワークエッジで OpenShift Container Platform クラスターを設定およびデプロイする

法律上の通知

概要

このドキュメントでは、GitOps ZTP を使用して OpenShift Container Platform クラスターを設定およびデプロイし、ネットワーク遠端のサイトをプロビジョニングおよび管理する方法について説明します。

目次

第1章 ネットワークファー遠端の課題

地理的に離れた場所にある多くのサイトを管理する場合、エッジコンピューティングには複雑な課題があります。GitOps Zero Touch Provisioning (ZTP) を使用して、ネットワークの遠端にあるサイトをプロビジョニングおよび管理します。

1.1. ネットワークファーエッジの課題を克服する

今日、サービスプロバイダーは、自社のインフラストラクチャーをネットワークのエッジにデプロイメントしたいと考えています。これには重大な課題があります。

- 多数のエッジサイトのデプロイメントを並行してどのように処理しますか?
- 切断された環境にサイトをデプロイメントする必要がある場合はどうなりますか?
- 大規模なクラスター群のライフサイクルをどのように管理していますか?

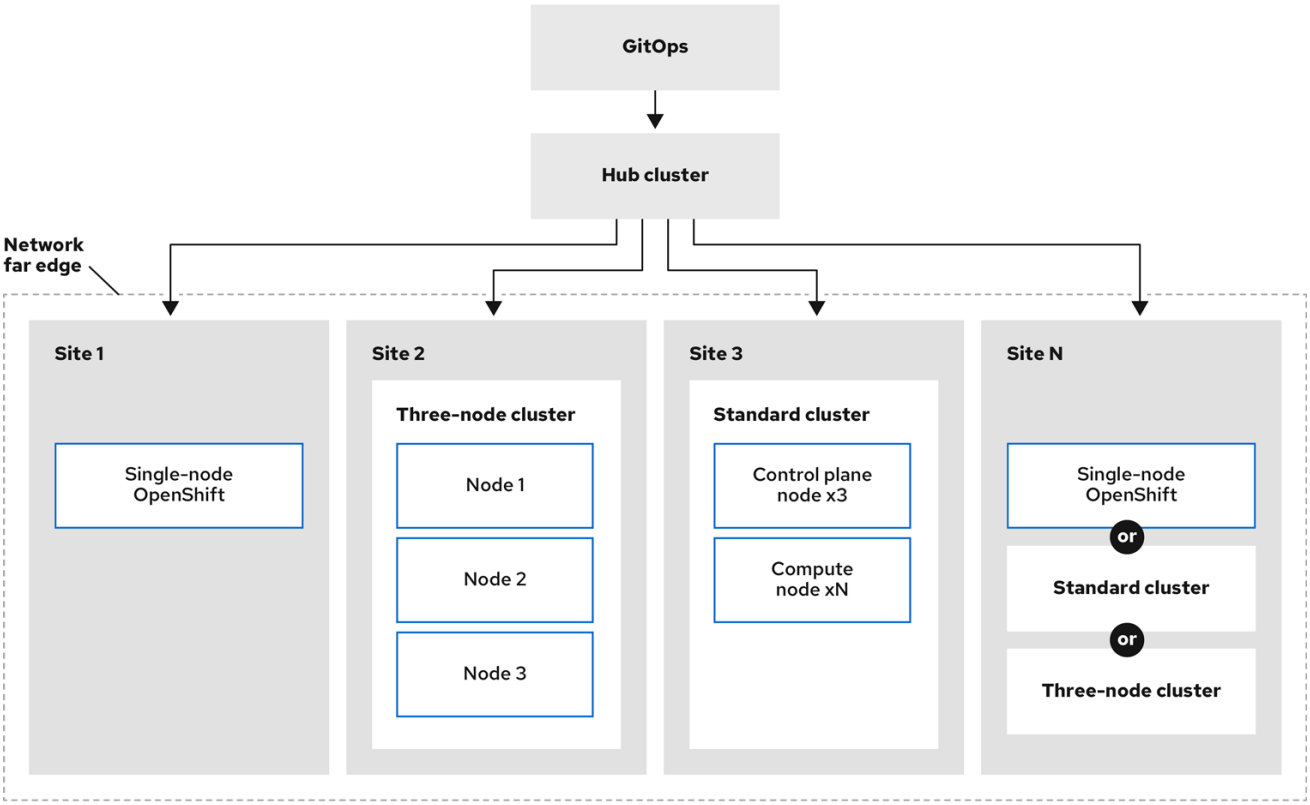
GitOps Zero Touch Provisioning (ZTP) と **GitOps** は、ベアメタル機器の宣言的なサイト定義と設定を使用してリモートエッジサイトを大規模にプロビジョニングできるようにすることで、これらの課題を解決します。テンプレートまたはオーバーレイ設定は、CNF ワークロードに必要な OpenShift Container Platform 機能をインストールします。インストールとアップグレードの全ライフサイクルは、GitOps ZTP パイプラインを通じて処理されます。

GitOps ZTP は、インフラストラクチャーのデプロイメントに GitOps を使用します。GitOps では、Git リポジトリーに格納されている宣言型 YAML ファイルとその他の定義済みパターンを使用します。Red Hat Advanced Cluster Management (RHACM) は、Git リポジトリーを使用してインフラストラクチャーのデプロイメントを推進します。

GitOps は、トレーサビリティ、ロールベースのアクセス制御 (RBAC)、および各サイトの望ましい状態に関する信頼できる唯一の情報源を提供します。スケーラビリティの問題は、Git の方法論と、Webhook を介したイベント駆動型操作によって対処されます。

GitOps ZTP パイプラインがエッジノードに配信する宣言的なサイト定義と設定のカスタムリソース (CR) を作成すると、GitOps ZTP ワークフローが開始します。

以下の図は、エッジサイトフレームワーク内で GitOps ZTP が機能する仕組みを示しています。



1.2. GITOPS ZTP を使用したネットワーク遠端でのクラスタープロビジョニング

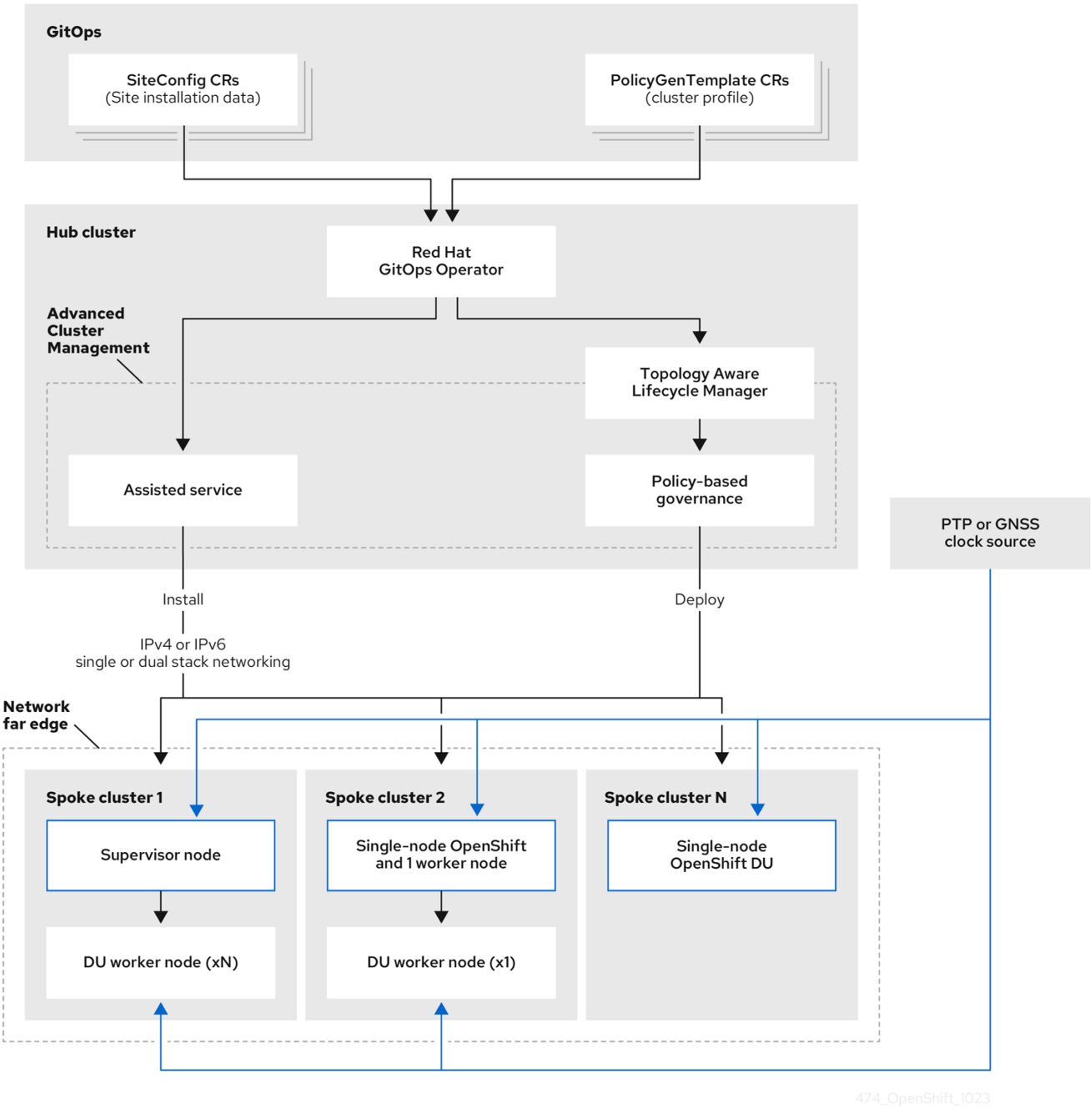
Red Hat Advanced Cluster Management (RHACM) は、単一のハブクラスターが多数のスポーククラスターを管理するハブアンドスポークアーキテクチャーでクラスターを管理します。RHACM を実行するハブクラスターは、GitOps Zero Touch Provisioning (ZTP) と、RHACM のインストール時にデプロイメントされるアシストサービスを使用して、マネージドクラスターのプロビジョニングおよびデプロイメントを実行します。

アシストサービスは、ベアメタルで実行される単一ノードクラスター、3 ノードクラスター、または標準クラスターで OpenShift Container Platform のプロビジョニングを処理します。

GitOps ZTP を使用して OpenShift Container Platform でベアメタルホストをプロビジョニングおよび維持する方法の概要は次のとおりです。

- RHACM を実行するハブクラスターは、OpenShift Container Platform リリースイメージをミラーリングする OpenShift イメージレジストリーを管理します。RHACM は、OpenShift イメージレジストリーを使用して、マネージドクラスターをプロビジョニングします。
- ベアメタルホストは、Git リポジトリーでバージョン管理された YAML 形式のインベントリーファイルで管理します。
- ホストをマネージドクラスターとしてプロビジョニングする準備を整え、RHACM とアシストサービスを使用してサイトにベアメタルホストをインストールします。

クラスターのインストールとデプロイメントは、最初のインストールフェーズと、その後の設定フェーズおよびデプロイメントフェーズを含む 2 段階のプロセスです。次の図は、このワークフローを示しています。



1.3. SITECONFIG リソースと RHACM を使用したマネージドクラスターのインストール

GitOps Zero Touch Provisioning (ZTP) は、Git リポジトリー内の **SiteConfig** カスタムリソース (CR) を使用して、OpenShift Container Platform クラスターのインストールプロセスを管理します。**SiteConfig** CR には、インストールに必要なクラスター固有のパラメーターが含まれています。ユーザー定義の追加マニフェストを含む、インストール中に選択した設定 CR を適用するためのオプションがあります。

ZTP GitOps プラグインは、**SiteConfig** CR を処理して、ハブクラスター上に CR コレクションを生成します。これにより、Red Hat Advanced Cluster Management (RHACM) のアシストサービスがトリガーされ、OpenShift Container Platform がベアメタルホストにインストールされます。ハブクラスターのこれらの CR で、インストールステータスとエラーメッセージを確認できます。

単一クラスターは、手動でプロビジョニングするか、GitOps ZTP を使用してバッチでプロビジョニングできます。

単一クラスターのプロビジョニング

単一の **SiteConfig** CR と、関連するインストールおよび設定 CR をクラスター用に作成し、それらをハブクラスターに適用して、クラスターのプロビジョニングを開始します。これは、より大きなスケールにデプロイする前に CR をテストするのに適した方法です。

多くのクラスターのプロビジョニング

Git リポジトリーで **SiteConfig** と関連する CR を定義することにより、最大 400 のバッチでマネージドクラスターをインストールします。ArgoCD は **SiteConfig** CR を使用してサイトをデプロイします。RHACM ポリシージェネレーターはマニフェストを作成し、それらをハブクラスターに適用します。これにより、クラスターのプロビジョニングプロセスが開始されます。

1.4. ポリシーと POLICYGENTEMPLATE リソースを使用したマネージドクラスターの設定

GitOps Zero Touch Provisioning (ZTP) は、Red Hat Advanced Cluster Management (RHACM) を使用して、設定を適用するためのポリシーベースのガバナンスアプローチを使用してクラスターを設定します。

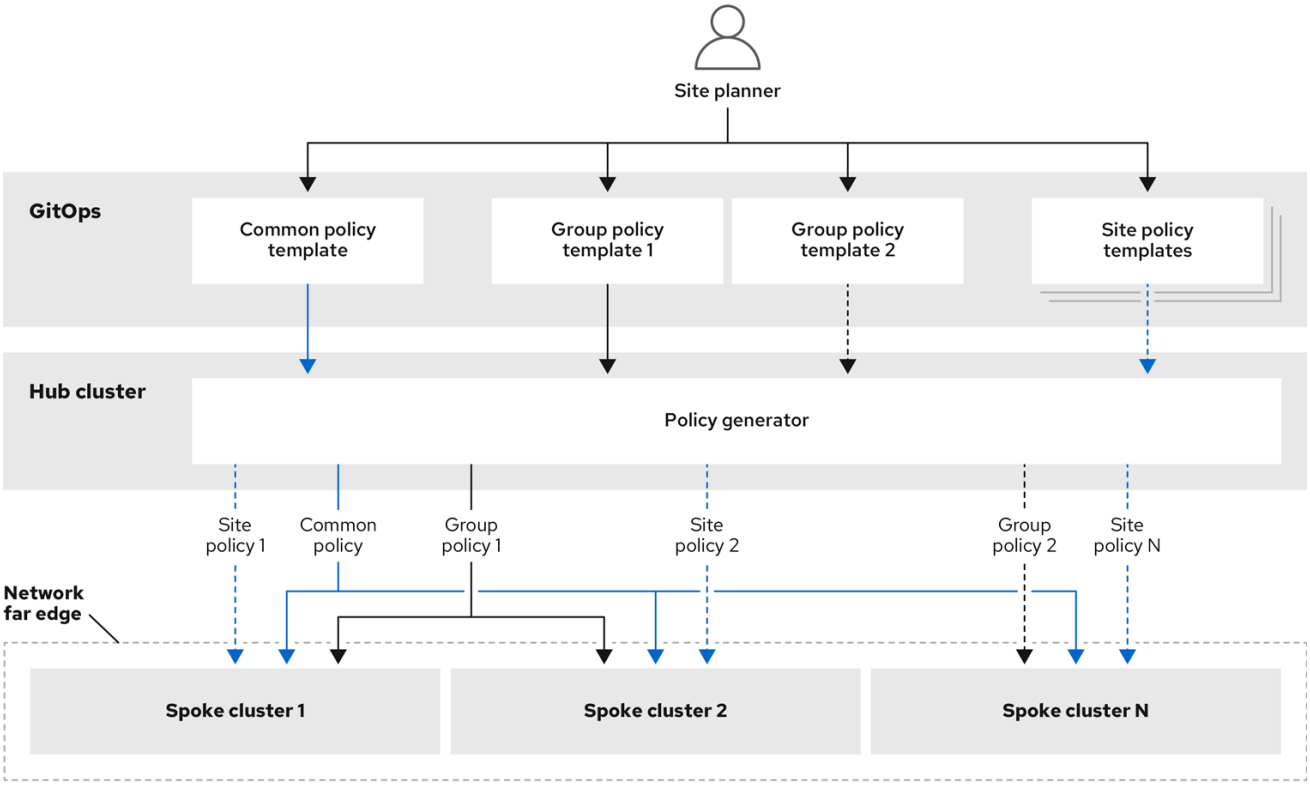
ポリシージェネレーターまたは **PolicyGen** は、簡潔なテンプレートから RHACM ポリシーを作成できるようにする GitOps Operator のプラグインです。このツールは、複数の CR を 1 つのポリシーに組み合わせることができ、フリート内のクラスターのさまざまなサブセットに適用される複数のポリシーを生成できます。

注記

スケーラビリティを確保し、クラスターのフリート全体で設定を管理する複雑さを軽減するには、できるだけ多くの共通性を持つ設定 CR を使用します。

- 可能であれば、フリート全体の共通ポリシーを使用して設定 CR を適用します。
- 次の優先事項は、クラスターの論理グループを作成して、グループポリシーの下で残りの設定を可能な限り管理することです。
- 設定が個々のサイトに固有のものである場合、ハブクラスターで RHACM テンプレートを使用して、サイト固有のデータを共通ポリシーまたはグループポリシーに挿入します。または、サイトに個別のサイトポリシーを適用します。

次の図は、ポリシージェネレーターがクラスターデプロイメントの設定フェーズで GitOps および RHACM と対話する方法を示しています。



クラスターの大規模なフリートの場合は、それらのクラスターの設定に高レベルの一貫性があるのが一般的です。

次の推奨されるポリシーの構造化では、設定 CR を組み合わせていくつかの目標を達成しています。

- 一般的な設定を一度説明すれば、フリートに適用できます。
- 維持および管理されるポリシーの数を最小限に抑えます。
- クラスターバリアントの一般的な設定の柔軟性をサポートします。

表1.1 推奨される PolicyGenTemplate ポリシーカテゴリー

ポリシーのカテゴリー	説明
共通	共通カテゴリーに存在するポリシーは、フリート内のすべてのクラスターに適用されます。共通の **PolicyGenTemplate** CR を使用して、すべてのクラスタータイプに共通のインストール設定を適用します。
グループ	groups カテゴリーに存在するポリシーは、フリート内のクラスターのグループに適用されます。グループ **PolicyGenTemplate** CR を使用して、単一ノード、3 ノード、および標準クラスターインストールの特定の側面を管理します。クラスターグループは、地理的地域、ハードウェアバリアントなどに従うこともできます。
サイト	sites カテゴリーに存在するポリシーが特定のクラスターに適用されます。どのクラスターでも、独自の特定のポリシーを維持できます。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および **PolicyGenerator** CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較
- GitOps ZTP Git リポジトリーの準備

第2章 GITOPS ZTP 用のハブクラスターの準備

切断された環境で RHACM を使用するには、OpenShift Container Platform リリースイメージと必要な Operator イメージを含む Operator Lifecycle Manager (OLM) カタログをミラーリングするミラーレジストリーを作成します。OLM は Operator およびそれらの依存関係をクラスターで管理し、インストールし、アップグレードします。切断されたミラーホストを使用して、ベアメタルホストのプロビジョニングに使用される RHCOS ISO および RootFS ディスクイメージを提供することもできます。

2.1. TELCO RAN DU 4.15 の検証済みソフトウェアコンポーネント

Red Hat Telco RAN DU 4.15 ソリューションは、次に示す OpenShift Container Platform のマネージドクラスターおよびハブクラスター用の Red Hat ソフトウェア製品を使用して検証されています。

表2.1 Telco RAN DU マネージドクラスターの検証済みソフトウェアコンポーネント

コンポーネント	ソフトウェアバージョン
マネージドクラスターのバージョン	4.16
Cluster Logging Operator	5.9
Local Storage Operator	4.16
PTP Operator	4.16
SRIOV Operator	4.16
Node Tuning Operator	4.16
Logging Operator	4.16
SRIOV-FEC Operator	2.9

表2.2 ハブクラスターの検証済みソフトウェアコンポーネント

コンポーネント	ソフトウェアバージョン
ハブクラスターのバージョン	4.16
GitOps ZTP プラグイン	4.16
Red Hat Advanced Cluster Management (RHACM)	2.11
Red Hat OpenShift GitOps	1.12
Topology Aware Lifecycle Manager (TALM)	4.16

2.2. GITOPS ZTP で推奨されるハブクラスター仕様とマネージドクラスターの制限

GitOps Zero Touch Provisioning (ZTP) を使用すると、地理的に分散した地域やネットワークにある数千のクラスターを管理できます。Red Hat Performance and Scale ラボは、ラボ環境内の単一の Red Hat Advanced Cluster Management (RHACM) ハブクラスターから、より小さな DU プロファイルを使用して 3,500 個の仮想シングルノード OpenShift クラスター作成および管理することに成功しました。

実際の状況では、管理できるクラスター数のスケーリング制限は、ハブクラスターに影響を与えるさまざまな要因によって異なります。以下に例を示します。

ハブクラスターのリソース

利用可能なハブクラスターのホストリソース (CPU、メモリー、ストレージ) は、ハブクラスターが管理できるクラスターの数を決定する重要な要素です。ハブクラスターに割り当てられるリソースが多いほど、対応できるマネージドクラスターの数も多くなります。

ハブクラスターストレージ

ハブクラスターホストのストレージ IOPS 評価と、ハブクラスターホストが NVMe ストレージを使用するかどうかは、ハブクラスターのパフォーマンスと管理できるクラスターの数に影響を与える可能性があります。

ネットワーク帯域幅と遅延

ハブクラスターとマネージドクラスター間のネットワーク接続が遅い、大きく遅延する場合、ハブクラスターによる複数クラスターの管理方法に影響を与える可能性があります。

マネージドクラスターのサイズと複雑さ

マネージドクラスターのサイズと複雑さも、ハブクラスターの容量に影響します。より多くのノード、namespace、リソースを備えた大規模なマネージドクラスターには、追加の処理リソースと管理リソースが必要です。同様に、RAN DU プロファイルや多様なワークロードなどの複雑な設定を持つクラスターは、ハブクラスターからより多くのリソースを必要とする可能性があります。

管理ポリシーの数

ハブクラスターによって管理されるポリシーの数は、それらのポリシーにバインドされているマネージドクラスターの数に対してスケーリングされており、これらは管理できるクラスターの数を決定する重要な要素です。

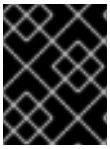
ワークロードのモニタリングと管理

RHACM は、マネージドクラスターを継続的にモニタリングおよび管理します。ハブクラスター上で実行されるモニタリングおよび管理ワークロードの数と複雑さは、ハブクラスターの容量に影響を与える可能性があります。集中的なモニタリングや頻繁な調整操作には追加のリソースが必要となる場合があり、管理可能なクラスターの数が制限される可能性があります。

RHACM のバージョンと設定

RHACM のバージョンが異なると、パフォーマンス特性やリソース要件も異なる場合があります。さらに、同時リコンシリエーションの数やヘルスチェックの頻度などの RHACM 設定は、ハブクラスターのマネージドクラスター容量に影響を与える可能性があります。

次の代表的な設定とネットワーク仕様を使用して、独自の Hub クラスターとネットワーク仕様を開発します。

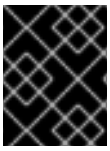


重要

次のガイドラインは、社内のラボのベンチマークテストのみに基づいており、完全なベアメタルホストの仕様を表すものではありません。

表2.3 代表的な 3 ノードハブクラスターマシンの仕様

要件	説明
OpenShift Container Platform	バージョン 4.13
RHACM	バージョン 2.7
Topology Aware Lifecycle Manager (TALM)	バージョン 4.13
サーバーハードウェア	Dell PowerEdge R650 ラックサーバー 3 台
NVMe ハードディスク	• **/var/lib/etcd** 用の 50 GB ディスク • **/var/lib/containers** 用の 2.9 TB ディスク
SSD ハードディスク	• 1 つの SSD を、15 の シンプロビジョニングされた 200 GB の論理ボリュームに分割して **PV** CR としてプロビジョニング。 • 非常に大規模な **PV** リソースとして機能する 1 つの SSD
適用された DU プロファイルポリシーの数	5



重要

次のネットワーク仕様は、典型的な実際の RAN ネットワークを表しており、テスト中にスケールラボ環境に適用されます。

表2.4 模擬ラボ環境のネットワーク仕様

仕様	説明
ラウンドトリップタイム (RTT) 遅延	50 ms
パケットロス	0.02% のパケットロス
ネットワーク帯域幅の制限	20 Mbps

関連情報

- RHACM を使用したシングルノード OpenShift クラスターの作成と管理

2.3. 切断された環境での GITOPS ZTP のインストール

切断された環境のハブクラスターで Red Hat Advanced Cluster Management (RHACM)、Red Hat OpenShift GitOps、Topology Aware Lifecycle Manager (TALM) を使用して、複数のマネージドクラスターのデプロイを管理します。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- クラスターで使用するために、切断されたミラーレジストリーを設定しました。

注記

作成する非接続ミラーレジストリーには、ハブクラスターで実行されている TALM のバージョンと一致する TALM バックアップおよび事前キャッシュイメージのバージョンが含まれている必要があります。スポーククラスターは、切断されたミラーレジストリーでこれらのイメージを解決できる必要があります。

手順

- ハブクラスターに RHACM をインストールします。非接続環境での RHACM のインストール を参照してください。
- ハブクラスターに GitOps と TALM をインストールします。

関連情報

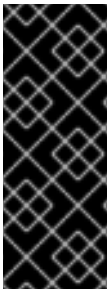
- OpenShift GitOps のインストール
- TALM のインストール
- Operator カタログのミラーリング

2.4. RHCOS ISO および ROOTFS イメージの非接続ミラーホストへの追加

Red Hat Advanced Cluster Management (RHACM) を使用して非接続環境にクラスターのインストールを開始する前に、最初に使用する Red Hat Enterprise Linux CoreOS (RHCOS) イメージをホストする必要があります。切断されたミラーを使用して RHCOS イメージをホストします。

前提条件

- ネットワーク上で RHCOS イメージリソースをホストするように HTTP サーバーをデプロイして設定します。お使いのコンピューターから HTTP サーバーにアクセスでき、作成するマシンからもアクセスできる必要があります。



重要

RHCOS イメージは OpenShift Container Platform の各リリースごとに変更されない可能性があります。インストールするバージョン以下の最新バージョンのイメージをダウンロードする必要があります。利用可能な場合は、OpenShift Container Platform バージョンに一致するイメージのバージョンを使用します。ホストに RHCOS をインストールするには、ISO および RootFS イメージが必要です。RHCOS QCOW2 イメージは、このインストールタイプではサポートされません。

手順

1. ミラーホストにログインします。

2. mirror.openshift.com から RHCOS ISO イメージおよび RootFS イメージを取得します。以下は例になります。

   a. 必要なイメージ名と OpenShift Container Platform のバージョンを環境変数としてエクスポートします。

   ```
   $ export ISO_IMAGE_NAME=<iso_image_name> 1
   ```

   ```
   $ export ROOTFS_IMAGE_NAME=<rootfs_image_name> 1
   ```

   ```
   $ export OCP_VERSION=<ocp_version> 1
   ```

   1 ISO イメージ名 (例: **rhcos-4.15.1-x86_64-live.x86_64.iso**)

   1 RootFS イメージ名 (例: **rhcos-4.15.1-x86_64-live-rootfs.x86_64.img**)

   1 OpenShift Container Platform バージョン (例: **4.15.1**)

   b. 必要なイメージをダウンロードします。

   ```
   $ sudo wget https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.16/${OCP_VERSION}/${ISO_IMAGE_NAME} -O /var/www/html/${ISO_IMAGE_NAME}
   ```

   ```
   $ sudo wget https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.16/${OCP_VERSION}/${ROOTFS_IMAGE_NAME} -O /var/www/html/${ROOTFS_IMAGE_NAME}
   ```

検証手順

- イメージが正常にダウンロードされ、非接続ミラーホストで提供されることを確認します。以下に例を示します。

  ```
  $ wget http://$(hostname)/${ISO_IMAGE_NAME}
  ```

  出力例

  ```
  Saving to: rhcos-4.16.1-x86_64-live.x86_64.iso
  rhcos-4.16.1-x86_64-live.x86_64.iso-  11%[====>    ]  10.01M  4.71MB/s
  ```

関連情報

- ミラーレジストリーの作成
- 非接続インストールのイメージのミラーリング

2.5. 支援サービスの有効化

Red Hat Advanced Cluster Management (RHACM) は、アシストサービスを使用して OpenShift Container Platform クラスターをデプロイします。Red Hat Advanced Cluster Management (RHACM) で MultiClusterHub Operator を有効にすると、支援サービスが自動的にデプロイされます。その後、す

べての napespace を監視し、ミラーレジストリー HTTP サーバーでホストされている ISO および RootFS イメージへの参照を使用して、**AgentServiceConfig** カスタムリソース (CR) を更新するように **Provisioning** リソースを設定する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- RHACM で MultiClusterHub が有効になっている。

手順

1. **Provisioning** リソースを有効にして、すべての namespace を監視し、非接続環境のミラーを設定します。詳細は、Central Infrastructure Management サービスの有効化 を参照してください。
2. 以下のコマンドを実行して、**AgentServiceConfig** CR を更新します。

```
$ oc edit AgentServiceConfig
```

3. CR の **items.spec.osImages** フィールドに次のエントリーを追加します。

```
- cpuArchitecture: x86_64
  openshiftVersion: "4.16"
  rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
  url: https://<mirror-registry>/<path>/rhcos-live.x86_64.iso
```

ここでは、以下のようになります。

<host>
ターゲットミラーレジストリー HTTP サーバーの完全修飾ドメイン名 (FQDN) です。

<path>
ターゲットミラーレジストリー上のイメージへのパスです。

エディターを保存して終了し、変更を適用します。

2.6. 切断されたミラーレジストリーを使用するためのハブクラスターの設定

切断された環境で切断されたミラーレジストリーを使用するようにハブクラスターを設定できます。

前提条件

- Red Hat Advanced Cluster Management (RHACM) 2.9 をインストール済みの非接続ハブクラスターのインストールがある。
- HTTP サーバーで **rootfs** および **iso** イメージをホストしている。**OpenShift Container Platform イメージリポジトリーのミラーリング** に関するガイダンスについては、**関連情報** セクションを参照してください。

警告

HTTP サーバーに対して TLS を有効にする場合、ルート証明書がクライアントによって信頼された機関によって署名されていることを確認し、OpenShift Container Platform ハブおよびマネージドクラスターと HTTP サーバー間の信頼された証明書チェーンを検証する必要があります。信頼されていない証明書で設定されたサーバーを使用すると、イメージがイメージ作成サービスにダウンロードされなくなります。信頼されていない HTTPS サーバーの使用はサポートされていません。

手順

1. ミラーレジストリー設定を含む **ConfigMap** を作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: assisted-installer-mirror-config
  namespace: multicluster-engine 1
  labels:
    app: assisted-service
data:
  ca-bundle.crt: | 2
    -----BEGIN CERTIFICATE-----
    <certificate_contents>
    -----END CERTIFICATE-----

  registries.conf: | 3
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

    [[registry]]
      prefix = ""
      location = "quay.io/example-repository" 4
      mirror-by-digest-only = true

      [[registry.mirror]]
      location = "mirror1.registry.corp.com:5000/example-repository" 5
```

1. **ConfigMap** namespace は **multicluster-engine** に設定する必要があります。
2. ミラーレジストリーの作成時に使用されるミラーレジストリーの証明書。
3. ミラーレジストリーの設定ファイル。ミラーレジストリー設定は、検出イメージの **/etc/containers/registries.conf** ファイルにミラー情報を追加します。ミラー情報は、インストールプログラムに渡される際、**install-config.yaml** ファイルの **imageContentSources** セクションに保存されます。ハブクラスターで実行される Assisted Service Pod は、設定されたミラーレジストリーからコンテナーイメージをフェッチします。
4. ミラーレジストリーの URL。ミラーレジストリーを設定する場合は、**oc adm release Mirror** コマンドを実行して、**imageContentSources** セクションの URL を使用する必要があります。詳細は、**OpenShift Container Platform イメージリポジトリーのミラーリン**

グ セクションを参照してください。

5 **registries.conf** ファイルで定義されるレジストリーは、レジストリーではなくリポジトリーによってスコープが指定される必要があります。この例では、**quay.io/example-repository** リポジトリーと **mirror1.registry.corp.com:5000/example-repository** リポジトリーの両方のスコープが **example-repository** リポジトリーにより指定されます。

これにより、以下のように **AgentServiceConfig** カスタムリソースの **mirrorRegistryRef** が更新されます。

出力例

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  namespace: multicluster-engine 1
spec:
  databaseStorage:
    volumeName: <db_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <db_storage_size>
  filesystemStorage:
    volumeName: <fs_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <fs_storage_size>
  mirrorRegistryRef:
    name: assisted-installer-mirror-config 2
  osImages:
    - openshiftVersion: <ocp_version>
      url: <iso_url> 3
```

1 **ConfigMap** namespace と一致するように、**AgentServiceConfig** namespace を **multicluster-engine** に設定します。

2 関連する **ConfigMap** CR で指定された定義と一致するように、**mirrorRegistryRef.name** を設定します。

3 **httpd** サーバーでホストされる ISO の URL を設定します。

重要

クラスターのインストール時には、有効な NTP サーバーが必要です。適切な NTP サーバーが使用可能であり、切断されたネットワークを介してインストール済みクラスターからアクセスできることを確認してください。

関連情報

- OpenShift Container Platform イメージリポジトリーのミラーリング

2.7. 非認証レジストリーを使用するためのハブクラスターの設定

非認証レジストリーを使用するようにハブクラスターを設定できます。非認証レジストリーは、イメージへのアクセスとダウンロードに認証を必要としません。

前提条件

- ハブクラスターがインストールおよび設定され、ハブクラスターに Red Hat Advanced Cluster Management (RHACM) がインストールされている。
- OpenShift Container Platform CLI (oc) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで使用するために非認証レジストリーを設定している。

手順

1. 次のコマンドを実行して、**AgentServiceConfig** カスタムリソース (CR) を更新します。

```
$ oc edit AgentServiceConfig agent
```

2. CR に **unauthenticatedRegistries** フィールドを追加します。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
spec:
  unauthenticatedRegistries:
  - example.registry.com
  - example.registry2.com
  ...
```

非認証レジストリーは、**AgentServiceConfig** リソースの **spec.unauthenticatedRegistries** の下に一覧表示されます。このリストにあるレジストリーのエントリーは、スポーククラスターのインストールに使用されるプルシークレットに含める必要はありません。**assisted-service** は、インストールに使用されるすべてのイメージレジストリーの認証情報がプルシークレットに含まれていることを確認して、プルシークレットを検証します。

注記

ミラーレジストリーは自動的に無視リストに追加されるため、**spec.unauthenticatedRegistries** の下に追加する必要はありません。**ConfigMap** で **PUBLIC_CONTAINER_REGISTRIES** 環境変数を指定すると、デフォルト値が指定した値でオーバーライドされます。**PUBLIC_CONTAINER_REGISTRIES** のデフォルトは quay.io および registry.svc.ci.openshift.org です。

検証

次のコマンドを実行して、ハブクラスターから新しく追加されたレジストリーにアクセスできることを確認します。

1. ハブクラスターへのデバッグシェルプロンプトを開きます。

```
$ oc debug node/<node_name>
```

2. 次のコマンドを実行して、非認証レジストリーへのアクセスをテストします。

```
sh-4.4# podman login -u kubeadmin -p $(oc whoami -t) <unauthenticated_registry>
```

ここでは、以下のようになります。

<unauthenticated_registry>

unauthenticated-image-registry.openshift-image-registry.svc:5000 などの新しいレジストリーです。

出力例

```
Login Succeeded!
```

2.8. ARGOCD を使用したハブクラスターの設定

GitOps Zero Touch Provisioning (ZTP) を使用して、サイトごとに必要なインストールおよびポリシーカスタムリソース (CR) を生成する一連の ArgoCD アプリケーションでハブクラスターを設定できます。

注記

Red Hat Advanced Cluster Management (RHACM) は **SiteConfig** CR を使用して、ArgoCD の Day 1 マネージドクラスターインストール CR を生成します。各 ArgoCD アプリケーションは、最大 300 個の **SiteConfig** CR を管理できます。

前提条件

- Red Hat Advanced Cluster Management (RHACM) と Red Hat OpenShift GitOps がインストールされた OpenShift Container Platform ハブクラスターがあります。
- 「GitOps ZTP サイト設定リポジトリーの準備」セクションで説明されているように、GitOps ZTP プラグインコンテナーから参照デプロイメントを抽出しました。参照デプロイメントを抽出すると、次の手順で参照される **out/argocd/deployment** ディレクトリーが作成されます。

手順

1. ArgoCD パイプライン設定を準備します。
   a. example ディレクトリーと同様にディレクトリー構造で Git リポジトリーを作成します。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。
   b. ArgoCD UI を使用して、リポジトリーへのアクセスを設定します。**Settings** で以下を設定します。
      - **リポジトリー**: 接続情報を追加します。URL は **.git** などで終わって いる必要があります。**https://repo.example.com/repo.git** とクレデンシャルを指定します。
      - **certificates**: 必要に応じて、リポジトリーのパブリック証明書を追加します。

c. 2 つの ArgoCD アプリケーション、**out/argocd/deployment/clusters-app.yaml** と **out/argocd/deployment/policies-app.yaml** を、Git リポジトリーに基づいて修正します。

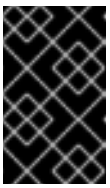
- Git リポジトリーを参照するように URL を更新します。URL は **.git** で終わります (例: **https://repo.example.com/repo.git**)。
- **targetRevision** は、監視する Git リポジトリーブランチを示します。
- **path** は、それぞれ **SiteConfig** CR および **PolicyGenerator** または **PolicyGentemplate** CR へのパスを指定します。

2. GitOps ZTP プラグインをインストールするには、ハブクラスター内の ArgoCD インスタンスに、関連するマルチクラスターエンジン (MCE) サブスクリプションイメージをパッチ適用します。以前に **out/argocd/deployment/** ディレクトリーに展開したパッチファイルを環境に合わせてカスタマイズします。

a. RHACM バージョンに一致する **multicluster-operators-subscription** イメージを選択します。

表2.5 multicluster-operators-subscription イメージバージョン

OpenShift Container Platform バージョン	RHACM バージョン	MCE バージョン	MCE RHEL バージョン	MCE イメージ
4.14、4.15、4.16	2.8、2.9	2.8、2.9	RHEL 8	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.8** **registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.9**
4.14、4.15、4.16	2.10	2.10	RHEL 9	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10**



重要

multicluster-operators-subscription イメージのバージョンは、RHACM バージョンと一致する必要があります。MCE 2.10 リリース以降、RHEL 9 は **multicluster-operators-subscription** イメージのベースイメージです。

b. **out/argocd/deployment/argocd-openshift-gitops-patch.json** ファイルに次の設定を追加します。

```
{
  "args": [
    "-c",
    "mkdir -p /.config/kustomize/plugin/policy.open-cluster-management.io/v1/policygenerator && cp /policy-generator/PolicyGenerator-not-fips-compliant /.config/kustomize/plugin/policy.open-cluster-
```

```
management.io/v1/policygenerator/PolicyGenerator" (1)
 ],
 "command": [
  "/bin/bash"
 ],
 "image": "registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10", (2)
(3)
 "name": "policy-generator-install",
 "imagePullPolicy": "Always",
 "volumeMounts": [
  {
   "mountPath": "/.config",
   "name": "kustomize"
  }
 ]
}
```

(1) オプション: RHEL 9 イメージの場合、ArgoCD バージョンの **/policy-generator/PolicyGenerator-not-fips-compliant** フォルダーに必要なユニバーサル実行可能ファイルをコピーします。

(2) **multicluster-operators-subscription** イメージを RHACM バージョンに一致させます。

(3) 非接続環境では、**multicluster-operators-subscription** イメージの URL を、ご使用の環境の非接続レジストリーに相当するものに置き換えます。

c. ArgoCD インスタンスにパッチを適用します。以下のコマンドを実行します。

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

3. RHACM 2.7 以降では、マルチクラスターエンジンはデフォルトで **cluster-proxy-addon** 機能を有効にします。次のパッチを適用して、**cluster-proxy-addon** 機能を無効にし、このアドオンに関連するハブクラスターとマネージド Pod を削除します。以下のコマンドを実行します。

```
$ oc patch multiclusterengines.multicluster.openshift.io multiclusterengine --type=merge --
patch-file out/argocd/deployment/disable-cluster-proxy-addon.json
```

4. 次のコマンドを実行して、パイプライン設定をハブクラスターに適用します。

```
$ oc apply -k out/argocd/deployment
```

2.9. GITOPS ZTP サイト設定リポジトリーの準備

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用する前に、サイト設定データをホストする Git リポジトリーを準備する必要があります。

前提条件

- 必要なインストールおよびポリシーのカスタムリソース (CR) を生成するためのハブクラスター GitOps アプリケーションを設定している。

- GitOps ZTP を使用してマネージドクラスターをデプロイしている。

手順

1. **SiteConfig** および **PolicyGenerator** または **PolicyGentemplate** CR の個別のパスでディレクトリー構造を作成します。

注記

SiteConfig および **PolicyGenerator** または **PolicyGentemplate** CR を個別のディレクトリーに保持します。SiteConfig ディレクトリーおよび PolicyGenTemplate ディレクトリーには、そのディレクトリー内のファイルを明示的に含める kustomization.yaml ファイルが含まれている必要があります。

2. 以下のコマンドを使用して **ztp-site-generate** コンテナーイメージから **argocd** ディレクトリーをエクスポートします。

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16
```

```
$ mkdir -p ./out
```

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./out
```

3. **out** ディレクトリーに以下のサブディレクトリーが含まれていることを確認します。

- **out/extra-manifest** には、**SiteConfig** が追加の manifest **configMap** の生成に使用するソース CR ファイルが含まれます。
- **out/source-crs** には、**PolicyGenTemplate が** Red Hat Advanced Cluster Management (RHACM) ポリシーを生成するために使用するソース CR ファイルが含まれています。
- **out/argocd/deployment** には、この手順の次のステップで使用するハブクラスターに適用するパッチおよび YAML ファイルが含まれます。
- **out/argocd/example** には、推奨の設定を表す **SiteConfig** ファイルおよび **PolicyGenTemplate** ファイルのサンプルが含まれています。

4. **out/source-crs** フォルダーおよびコンテンツを **PolicyGenerator** または **PolicyGentemplate** ディレクトリーにコピーします。

5. out/extra-manifests ディレクトリーには、RAN DU クラスターの参照マニフェストが含まれています。**out/extra-manifests** ディレクトリーを **SiteConfig** フォルダーにコピーします。このディレクトリーには、**ztp-site-generate** コンテナーからの CR のみを含める必要があります。ユーザー提供の CR をここに追加しないでください。ユーザー提供の CR を使用する場合は、そのコンテンツ用に別のディレクトリーを作成する必要があります。以下に例を示します。

```
example/
  ├── acmpolicygenerator
  │   ├── kustomization.yaml
  │   └── source-crs/
  ├── policygentemplates (1)
  │   ├── kustomization.yaml
  │   └── source-crs/
```

```
└── siteconfig
    ├── extra-manifests
    └── kustomization.yaml
```

1. **PolicyGenTemplate** CR を使用してクラスターを管理し、デプロイすることは、将来の OpenShift Container Platform リリースで非推奨になりました。同等の機能と改善機能は、Red Hat Advanced Cluster Management (RHACM)と **PolicyGenerator** CR を使用することで利用できます。

6. ディレクトリー構造と **kustomization.yaml** ファイルをコミットし、Git リポジトリーにプッシュします。Git への最初のプッシュには、**kustomization.yaml** ファイルが含まれている必要があります。

out/argocd/example のディレクトリー構造は、Git リポジトリーの構造およびコンテンツの参照として使用します。この構造には、単一ノード、3 ノード、標準クラスターの SiteConfig および **PolicyGenTemplate** の参照 CR が含まれます。使用されていないクラスタータイプの参照を削除します。

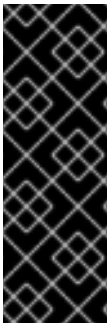
すべてのクラスタータイプについて、次のことを行う必要があります。

- **source-crs** サブディレクトリーを **acmpolicygenerator** または **policygentemplates** ディレクトリーに追加します。
- **extra-manifests** ディレクトリーを **siteconfig** ディレクトリーに追加します。

以下の例では、単一ノードクラスターのネットワークの CR のセットについて説明しています。

```
example/
  ├── acmpolicygenerator
  │   ├── acm-common-ranGen.yaml
  │   ├── acm-example-sno-site.yaml
  │   ├── acm-group-du-sno-ranGen.yaml
  │   ├── group-du-sno-validator-ranGen.yaml
  │   ├── kustomization.yaml
  │   ├── source-crs/
  │   └── ns.yaml
  └── siteconfig
      ├── example-sno.yaml
      ├── extra-manifests/ 1
      ├── custom-manifests/ 2
      ├── KlusterletAddonConfigOverride.yaml
      └── kustomization.yaml
```

1. **ztp-container** からの参照マニフェストが含まれます。
2. カスタムマニフェストが含まれます。



重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および **PolicyGenerator** CR を使用して利用できます。

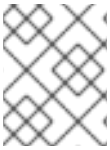
PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

2.10. バージョンに依存しないように GITOPS ZTP サイト設定リポジトリーを準備する

GitOps ZTP を使用して、OpenShift Container Platform のさまざまなバージョンを実行しているマネージドクラスターのソースカスタムリソース (CR) を管理できます。これは、ハブクラスター上で実行している OpenShift Container Platform のバージョンが、マネージドクラスター上で実行しているバージョンから独立している可能性があることを意味します。



注記

以下の手順は、クラスターポリシー管理に **PolicyGentemplate** リソースの代わりに **PolicyGenerator** リソースを使用していることを前提としています。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SiteConfig** CR と **PolicyGenTemplate** CR の個別のパスを持つディレクトリー構造を作成します。
2. **PolicyGenTemplate** ディレクトリー内に、使用可能にする OpenShift Container Platform バージョンごとにディレクトリーを作成します。バージョンごとに、次のリソースを作成します。
   - そのディレクトリー内のファイルを明示的に含む **kustomization.yaml** ファイル
   - **source-crs** ディレクトリーには、**ztp-site-generate** コンテナーからの参照 CR 設定ファイルが含まれます。
     ユーザー提供の CR を使用する場合は、CR 用に別のディレクトリーを作成する必要があります。
3. **/siteconfig** ディレクトリーに、使用可能にする OpenShift Container Platform バージョンごとにサブディレクトリーを作成します。バージョンごとに、コンテナーからコピーされる参照 CR 用のディレクトリーを少なくとも 1 つ作成します。ディレクトリーの名前や参照ディレクトリーの数に制限はありません。カスタムマニフェストを使用する場合は、個別のディレクトリーを作成する必要があります。

次の例では、OpenShift Container Platform のさまざまなバージョンのユーザー提供のマニフェストと CR を使用した構造について説明します。

```
├── acmpolicygenerator
│   ├── kustomization.yaml 1
│   ├── version_4.13 2
│   │   ├── common-ranGen.yaml
│   │   ├── group-du-sno-ranGen.yaml
│   │   ├── group-du-sno-validator-ranGen.yaml
│   │   ├── helix56-v413.yaml
│   │   ├── kustomization.yaml 3
│   │   ├── ns.yaml
│   │   └── source-crs/ 4
│   │       └── reference-crs/ 5
│   │       └── custom-crs/ 6
│   └── version_4.14 7
│       ├── common-ranGen.yaml
│       ├── group-du-sno-ranGen.yaml
│       ├── group-du-sno-validator-ranGen.yaml
│       ├── helix56-v414.yaml
│       ├── kustomization.yaml 8
│       ├── ns.yaml
│       └── source-crs/ 9
│        └── reference-crs/ 10
│        └── custom-crs/ 11
└── siteconfig
    ├── kustomization.yaml
    ├── version_4.13
    │   ├── helix56-v413.yaml
    │   ├── kustomization.yaml
    │   ├── extra-manifest/ 12
    │   └── custom-manifest/ 13
    └── version_4.14
        ├── helix57-v414.yaml
        ├── kustomization.yaml
        ├── extra-manifest/ 14
        └── custom-manifest/ 15
```

1 最上位の **kustomization** YAML ファイルを作成します。

2 7 カスタムの /**acmpolicygenerator** ディレクトリー内にバージョン固有のディレクトリーを作成します。

3 8 バージョンごとに **kustomization.yaml** ファイルを作成します。

4 9 **ztp-site-generate** コンテナーからの参照 CR を含めるために、バージョンごとに **source-crs** ディレクトリーを作成します。

5 10 ZTP コンテナーから展開されるポリシー CR の **reference-crs** ディレクトリーを作成します。

6 11 オプション: ユーザー提供の CR 用に **custom-crs** ディレクトリーを作成します。

12 14 カスタム /**siteconfig** ディレクトリー内にディレクトリーを作成し、**ztp-site-generate** コンテナーからの追加のマニフェストを含めます

コンテナーからの追加のマニフェストを含めます。

13 15 ユーザーによって提供されるマニフェストを保持するフォルダーを作成します。

注記

前の例では、カスタム **/siteconfig** ディレクトリー内の各バージョンサブディレクトリーにはさらに 2 つのサブディレクトリーが含まれており、1 つはコンテナーからコピーされた参照マニフェストを含み、もう 1 つは提供するカスタムマニフェスト用です。これらのディレクトリーに割り当てられた名前は一例です。ユーザー提供の CR を使用する場合は、**SiteConfig** CR の **extraManifests.searchPaths** の下にリストされている最後のディレクトリーが、ユーザー提供の CR を含むディレクトリーである必要があります。

4. **SiteConfig** CR を編集して、作成したディレクトリーの検索パスを含めます。**extraManifests.searchPaths** の下にリストされる最初のディレクトリーは、参照マニフェストを含むディレクトリーである必要があります。ディレクトリーがリストされている順序を考慮してください。ディレクトリーに同じ名前のファイルが含まれている場合は、最後のディレクトリーにあるファイルが優先されます。

SiteConfig CR の例

```
extraManifests:
  searchPaths:
  - extra-manifest/ 1
  - custom-manifest/ 2
```

1 参照マニフェストを含むディレクトリーは、**extraManifests.searchPaths** の下に最初にリストされる必要があります。

2 ユーザー提供の CR を使用している場合は、**SiteConfig** CR の **extraManifests.searchPaths** の下にリストされている最後のディレクトリーが、ユーザー提供の CR を含むディレクトリーである必要があります。

5. トップレベルの **kustomization.yaml** ファイルを編集して、アクティブな OpenShift Container Platform バージョンを制御します。以下は、最上位レベルの **kustomization.yaml** ファイルの例です。

```
resources:
- version_4.13 1
#- version_4.14 2
```

1 バージョン 4.13 をアクティブ化します。

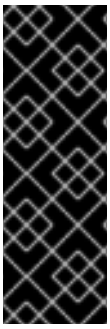
2 コメントを使用してバージョンを非アクティブ化します。

第3章 GITOPS ZTP の更新

GitOps Zero Touch Provisioning (ZTP) インフラストラクチャーは、ハブクラスター、Red Hat Advanced Cluster Management (RHACM)、およびOpenShift Container Platform マネージドクラスターとは別に更新できます。

注記

新しいバージョンが利用可能になったら、Red Hat OpenShift GitOps Operator を更新できます。GitOps ZTP プラグインを更新するときは、参照設定で更新されたファイルを確認し、変更が要件を満たしていることを確認してください。



重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および **PolicyGenerator** CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

3.1. GITOPS ZTP 更新プロセスの概要

以前のバージョンの GitOps ZTP インフラストラクチャーを実行している、完全に機能するハブクラスターの GitOps Zero Touch Provisioning (ZTP) を更新できます。更新プロセスにより、マネージドクラスターへの影響が回避されます。

注記

推奨コンテンツの追加など、ポリシー設定を変更すると、更新されたポリシーが作成され、マネージドクラスターにロールアウトして調整する必要があります。

GitOps ZTP インフラストラクチャーを更新するための戦略の概要は次のとおりです。

1. 既存のすべてのクラスターに **ztp-done** ラベルを付けます。
2. ArgoCD アプリケーションを停止します。
3. 新しい GitOps ZTP ツールをインストールします。
4. Git リポジトリーで必要なコンテンツおよびオプションの変更を更新します。
5. アプリケーション設定を更新して再起動します。

3.2. アップグレードの準備

次の手順を使用して、GitOps Zero Touch Provisioning (ZTP) アップグレードのためにサイトを準備します。

手順

1. GitOps ZTP で使用するために Red Hat OpenShift GitOps を設定するために使用されるカスタムリソース (CR) を持つ GitOps ZTP コンテナーの最新バージョンを取得します。

2. 次のコマンドを使用して、**argocd/deployment** ディレクトリーを抽出します。

```
$ mkdir -p ./update
```

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./update
```

   /update ディレクトリーには、次のサブディレクトリーが含まれています。

   - **update/extra-manifest**: **SiteConfig** CR が追加のマニフェスト **configMap** を生成するために使用するソース CR ファイルが含まれています。
   - **update/source-crs** には、**PolicyGenTemplate** CR が Red Hat Advanced Cluster Management (RHACM) ポリシーを生成するために使用するソース CR ファイルが含まれています。
   - **update/argocd/deployment** には、この手順の次のステップで使用するハブクラスターに適用するパッチおよび YAML ファイルが含まれます。
   - **update/argocd/example**: 推奨される設定を表す **SiteConfig** および **PolicyGenTemplate** ファイルの例が含まれています。

3. **clusters-app.yaml** ファイルおよび **policies-app.yaml** ファイルを更新して、Git リポジトリーのアプリケーションおよび URL、ブランチ、およびパスを反映します。
   アップグレードにポリシーの廃止につながる変更が含まれている場合は、アップグレードを実行する前に、廃止されたポリシーを削除する必要があります。

4. **/update** フォルダー内の設定およびデプロイソース CR と、フリートサイト CR を管理する Git リポジトリーとの間の変更を比較します。必要な変更をサイトリポジトリーに適用してプッシュします。

重要

GitOps ZTP を最新バージョンに更新するときは、**update/argocd/deployment** ディレクトリーからサイトリポジトリーに変更を適用する必要があります。古いバージョンの **argocd/deployment/** ファイルは使用しないでください。

3.3. 既存クラスターのラベル付け

既存のクラスターがツールの更新の影響を受けないようにするには、既存のすべてのマネージドクラスターに **ztp-done** ラベルを付けます。

注記

この手順は、Topology Aware Lifecycle Manager (TALM) でプロビジョニングされていないクラスターを更新する場合にのみ適用されます。TALM でプロビジョニングするクラスターには、自動的に **ztp-done** というラベルが付けられます。

手順

1. **local-cluster!=true** など、GitOps Zero Touch Provisioning (ZTP) でデプロイされたマネージドクラスターを一覧表示するラベルセレクターを見つけます。

```
$ oc get managedcluster -l 'local-cluster!=true'
```

2. 結果のリストに、GItOps ZTP でデプロイされたすべてのマネージドクラスターが含まれていることを確認してから、そのセレクターを使用して **ztp-done** ラベルを追加します。

```
$ oc label managedcluster -l 'local-cluster!=true' ztp-done=
```

3.4. 既存の GITOPS ZTP アプリケーションの停止

既存のアプリケーションを削除すると、Git リポジトリー内の既存のコンテンツに対する変更は、ツールの新しいバージョンが利用可能になるまでロールアウトされません。

deployment ディレクトリーからのアプリケーションファイルを使用します。アプリケーションにカスタム名を使用した場合は、まずこれらのファイルの名前を更新します。

手順

1. **clusters** アプリケーションで非カスケード削除を実行して、生成されたすべてのリソースをそのまま残します。

```
$ oc delete -f update/argocd/deployment/clusters-app.yaml
```

2. **policies** アプリケーションでカスケード削除を実行して、以前のすべてのポリシーを削除します。

```
$ oc patch -f policies-app.yaml -p '{"metadata": {"finalizers": ["resources-
finalizer.argocd.argoproj.io"]}}' --type merge
```

```
$ oc delete -f update/argocd/deployment/policies-app.yaml
```

3.5. GIT リポジトリーに必要な変更

ztp-site-generate コンテナーを以前のリリースの GitOps Zero Touch Provisioning (ZTP) から 4.10 以降にアップグレードする場合は、Git リポジトリーのコンテンツに関する追加の要件があります。これらの変更を反映するには、リポジトリー内の既存のコンテンツを更新する必要があります。

注記

以下の手順は、クラスターポリシー管理に **PolicyGentemplate** リソースの代わりに **PolicyGenerator** リソースを使用していることを前提としています。

- **PolicyGenerator** ファイルに必要な変更を加えます。
  すべての **PolicyGenerator** ファイルは、**ztp** で始まる **Namespace** で作成する必要があります。これにより、GitOps ZTP アプリケーションは、Red Hat Advanced Cluster Management (RHACM) が内部でポリシーを管理する方法と競合することなく、GitOps ZTP によって生成されたポリシー CR を管理できるようになります。

- **kustomization.yaml** ファイルをリポジトリーに追加します。
  すべての **SiteConfig** および **PolicyGenTemplate** CR は、それぞれのディレクトリー ツリーの下にある **kustomization.yaml** ファイルに含める必要があります。以下に例を示します。

```
├── acmpolicygenerator
│   ├── site1-ns.yaml
│   ├── site1.yaml
│   ├── site2-ns.yaml
│   ├── site2.yaml
│   ├── common-ns.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen-ns.yaml
│   ├── group-du-sno-ranGen.yaml
│   └── kustomization.yaml
└── siteconfig
    ├── site1.yaml
    ├── site2.yaml
    └── kustomization.yaml
```

注記

generator セクションにリストされているファイルには、**SiteConfig** または **{policy-gen-cr}** CR のみが含まれている必要があります。既存の YAML ファイルに **Namespace** などの他の CR が含まれている場合、これらの他の CR を別のファイルに取り出して、**resources** セクションにリストする必要があります。

PolicyGenerator kustomization ファイルには、**generator** セクションにすべての **PolicyGenerator** YAML ファイルが含まれ、**resources** セクションに **Namespace** CR が含まれている必要があります。以下に例を示します。

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- acm-common-ranGen.yaml
- acm-group-du-sno-ranGen.yaml
- site1.yaml
- site2.yaml

resources:
- common-ns.yaml
- acm-group-du-sno-ranGen-ns.yaml
- site1-ns.yaml
- site2-ns.yaml
```

SiteConfig kustomization ファイルには、すべての **SiteConfig** YAML ファイルが **generator** セクションおよびリソースの他の CR に含まれている必要があります。

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- site1.yaml
- site2.yaml
```

- **pre-sync.yaml** ファイルおよび **post-sync.yaml** ファイルを削除します。
  OpenShift Container Platform 4.10 以降では、**pre-sync.yaml** および **post-sync.yaml** ファイルは不要になりました。**update/deployment/kustomization.yaml** CR は、ハブクラスターでのポリシーのデプロイを管理します。

注記

SiteConfig ツリーと **{policy-gen-cr}** ツリーの両方の下に、一連の **pre-sync.yaml** および **post-sync.yaml** ファイルがあります。

- 推奨される変更の確認および組み込み
  各リリースには、デプロイされたクラスターに適用される設定に推奨される追加の変更が含まれる場合があります。通常、これらの変更により、OpenShift プラットフォーム、追加機能、またはプラットフォームのチューニングが改善された CPU の使用率が低下します。

  ネットワーク内のクラスターのタイプに適用可能なリファレンス **SiteConfig** および **PolicyGenTemplate** CR を確認します。これらの例は、GitOps ZTP コンテナーから抽出した **argocd/example** ディレクトリーにあります。

3.6. 新規 GITOPS ZTP アプリケーションのインストール

展開した **argocd/deployment** ディレクトリーを使用し、アプリケーションがサイトの Git リポジトリーをポイントすることを確認してから、deployment ディレクトリーの完全なコンテンツを適用します。ディレクトリーのすべての内容を適用すると、アプリケーションに必要なすべてのリソースが正しく設定されます。

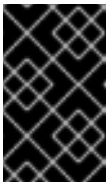
手順

1. GitOps ZTP プラグインをインストールするには、ハブクラスター内の ArgoCD インスタンスに、関連するマルチクラスターエンジン (MCE) サブスクリプションイメージをパッチ適用します。以前に **out/argocd/deployment/** ディレクトリーに展開したパッチファイルを環境に合わせてカスタマイズします。

   a. RHACM バージョンに一致する **multicluster-operators-subscription** イメージを選択します。

      表3.1 multicluster-operators-subscription イメージバージョン

OpenShift Container Platform バージョン	RHACM バージョン	MCE バージョン	MCE RHEL バージョン	MCE イメージ
4.14、4.15、4.16	2.8、2.9	2.8、2.9	RHEL 8	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.8** **registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v2.9**
4.14、4.15、4.16	2.10	2.10	RHEL 9	**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10**



重要

multicluster-operators-subscription イメージのバージョンは、RHACM バージョンと一致する必要があります。MCE 2.10 リリース以降、RHEL 9 は **multicluster-operators-subscription** イメージのベースイメージです。

b. **out/argocd/deployment/argocd-openshift-gitops-patch.json** ファイルに次の設定を追加します。

```
{
  "args": [
    "-c",
    "mkdir -p /.config/kustomize/plugin/policy.open-cluster-
management.io/v1/policygenerator && cp /policy-generator/PolicyGenerator-not-fips-
compliant /.config/kustomize/plugin/policy.open-cluster-
management.io/v1/policygenerator/PolicyGenerator" (1)
  ],
  "command": [
    "/bin/bash"
  ],
  "image": "registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v2.10", (2)
(3)
  "name": "policy-generator-install",
  "imagePullPolicy": "Always",
  "volumeMounts": [
    {
      "mountPath": "/.config",
      "name": "kustomize"
    }
  ]
}
```

1. オプション: RHEL 9 イメージの場合、ArgoCD バージョンの /**policy-generator/PolicyGenerator-not-fips-compliant** フォルダーに必要なユニバーサル実
2. **multicluster-operators-subscription** イメージを RHACM バージョンに一致させます。
3. 非接続環境では、**multicluster-operators-subscription** イメージの URL を、ご使用の環境の非接続レジストリーに相当するものに置き換えます。

c. ArgoCD インスタンスにパッチを適用します。以下のコマンドを実行します。

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

2. RHACM 2.7 以降では、マルチクラスターエンジンはデフォルトで **cluster-proxy-addon** 機能を有効にします。次のパッチを適用して、**cluster-proxy-addon** 機能を無効にし、このアドオンに関連するハブクラスターとマネージド Pod を削除します。以下のコマンドを実行します。

```
$ oc patch multiclusterengines.multicluster.openshift.io multiclusterengine --type=merge --patch-file out/argocd/deployment/disable-cluster-proxy-addon.json
```

3. 次のコマンドを実行して、パイプライン設定をハブクラスターに適用します。

```
$ oc apply -k out/argocd/deployment
```

3.7. GITOPS ZTP 設定の変更のロールアウト

推奨される変更を実装したために設定の変更がアップグレードに含まれていた場合、アップグレード プロセスの結果、ハブ クラスターの一連のポリシー CR が **Non-Compliant** 状態になります。GitOps Zero Touch Provisioning (ZTP) バージョン 4.10 以降の **ztp-site-generate** コンテナーの場合、これらのポリシーは **inform** モードに設定されており、ユーザーが追加の手順を実行しないとマネージドクラスターにプッシュされません。これにより、クラスターへの潜在的に破壊的な変更を、メンテナンス ウィンドウなどでいつ変更が行われたか、および同時に更新されるクラスターの数に関して管理できるようになります。

変更をロールアウトするには、TALM ドキュメントの詳細に従って、1 つ以上の **ClusterGroupUpgrade** CR を作成します。CR には、スポーク クラスターにプッシュする **Non-Compliant** ポリシーのリストと、更新に含めるクラスターのリストまたはセレクターが含まれている必要があります。

関連情報

- Topology Aware Lifecycle Manager (TALM) については、Topology Aware Lifecycle Manager 設定について を参照してください。
- **ClusterGroupUpgrade** CR の作成は、自動作成された ZTP の ClusterGroupUpgrade CR について を参照してください。

第4章 RHACM および SITECONFIG リソースを使用したマネージドクラスターのインストール

Red Hat Advanced Cluster Management (RHACM) を使用して OpenShift Container Platform クラスターを大規模にプロビジョニングするには、アシストサービスと、コア削減テクノロジーが有効になっている GitOps プラグインポリシージェネレーターを使用します。GitOps Zero Touch Provisioning (ZTP) パイプラインは、クラスターのインストールを実行します。GitOps ZTP は、非接続環境で使用できます。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および **PolicyGenerator** CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

4.1. GITOPS ZTP および TOPOLOGY AWARE LIFECYCLE MANAGER

GitOps Zero Touch Provisioning (ZTP) は、Git に格納されたマニフェストからインストールと設定の CR を生成します。これらのアーティファクトは、Red Hat Advanced Cluster Management (RHACM)、アシストサービス、および Topology Aware Lifecycle Manager (TALM) が CR を使用してマネージドクラスターをインストールおよび設定する中央ハブクラスターに適用されます。GitOps ZTP パイプラインの設定フェーズでは、TALM を使用してクラスターに対する設定 CR の適用のオーケストレーションを行います。GitOps ZTP と TALM の間には、いくつかの重要な統合ポイントがあります。

Inform ポリシー

デフォルトでは、GitOps ZTP は、**inform** の修復アクションですべてのポリシーを作成します。これらのポリシーにより、RHACM はポリシーに関連するクラスターのコンプライアンスステータスを報告しますが、必要な設定は適用されません。GitOps ZTP プロセスの中で OpenShift をインストールした後に、TALM は作成された **inform** ポリシーをステップスルーし、ターゲットのマネージドクラスターに適用します。これにより、設定がマネージドクラスターに適用されます。クラスターライフサイクルの GitOps ZTP フェーズ以外では、影響を受けるマネージドクラスターで変更をすぐにロールアウトするリスクなしに、ポリシーを変更できます。TALM を使用して、修正されたクラスターのタイミングとセットを制御できます。

ClusterGroupUpgrade CR の自動作成

新しくデプロイされたクラスターの初期設定を自動化するために、TALM はハブクラスター上のすべての **ManagedCluster** CR の状態を監視します。新規に作成された **ManagedCluster** CR を含む **ztp-done** ラベルを持たない **ManagedCluster** CR が適用されると、TALM は以下の特性で **ClusterGroupUpgrade** CR を自動的に作成します。

- **ClusterGroupUpgrade** CR が **ztp-install** namespace に作成され、有効にされます。
- **ClusterGroupUpgrade** CR の名前は **ManagedCluster** CR と同じになります。

- クラスターセレクターには、その **ManagedCluster** CR に関連付けられたクラスターのみが含まれます。
- 管理ポリシーのセットには、**ClusterGroupUpgrade** の作成時に RHACM がクラスターにバインドされているすべてのポリシーが含まれます。
- 事前キャッシュは無効です。
- タイムアウトを 4 時間 (240 分) に設定。

有効な **ClusterGroupUpgrade** の自動生成により、ユーザーの介入を必要としないゼロタッチのクラスター展開が可能になります。さらに、**ztp-done** ラベルのない **ManagedCluster** に対して **ClusterGroupUpgrade** CR が自動的に作成されるため、そのクラスターの **ClusterGroupUpgrade** CR を削除するだけで失敗した ZTP インストールを再開できます。

Waves

PolicyGenerator または **PolicyGentemplate** CR から生成される各ポリシーには、**ztp-deploy-wave** アノテーションが含まれています。このアノテーションは、そのポリシーに含まれる各 CR と同じアノテーションに基づいています。wave アノテーションは、自動生成された **ClusterGroupUpgrade** CR でポリシーを順序付けするために使用されます。wave アノテーションは、自動生成された **ClusterGroupUpgrade** CR 以外には使用されません。

注記

同じポリシーのすべての CR には **ztp-deploy-wave** アノテーションに同じ設定が必要です。各 CR のこのアノテーションのデフォルト値は、**PolicyGenerator** または **PolicyGentemplate** で上書きできます。ソース CR の wave アノテーションは、ポリシーの wave アノテーションを判別し、設定するために使用されます。このアノテーションは、実行時に生成されるポリシーに含まれるビルドされる各 CR から削除されます。

TALM は、wave アノテーションで指定された順序で設定ポリシーを適用します。TALM は、各ポリシーが準拠しているのを待ってから次のポリシーに移動します。各 CR の wave アノテーションは、それらの CR がクラスターに適用されるための前提条件を確実に考慮することが重要である。たとえば、Operator は Operator の設定前後にインストールする必要があります。同様に、Operator 用 **CatalogSource** は、Operator 用サブスクリプションの前または同時にウェーブにインストールする必要があります。各 CR のデフォルトの波動値は、これらの前提条件を考慮したものです。

複数の CR およびポリシーは同じアンブ番号を共有できます。ポリシーの数を少なくすることで、デプロイメントを高速化し、CPU 使用率を低減させることができます。多くの CR を比較的少なくするのがベストプラクティスです。

各ソース CR でデフォルトの wave 値を確認するには、**ztp-site-generate** コンテナーイメージからデプロイメントした **out/source-crs** ディレクトリーに対して以下のコマンドを実行します。

```
$ grep -r "ztp-deploy-wave" out/source-crs
```

フェーズラベル

ClusterGroupUpgrade CR は自動的に作成され、そこには GitOps ZTP プロセスの開始時と終了時に **ManagedCluster** CR をラベルでアノテートするディレクティブが含まれています。
インストール後に GitOps ZTP 設定が開始されると、**ManagedCluster** に **ztp-running** ラベルが適用されます。すべてのポリシーがクラスターに修復され、完全に準拠されると、TALM は **ztp-running** ラベルを削除し、**ztp-done** ラベルを適用します。

informDuValidator ポリシーを使用するデプロイメントでは、クラスターが完全にアプリケーションをデプロイするための準備が整った時点で **ztp-done** ラベルが適用されます。これには、GitOps ZTP が適用された設定 CR の調整および影響がすべて含まれます。**ztp-done** ラベルは、TALM による **ClusterGroupUpgrade** CR の自動作成に影響します。クラスターの最初の GitOps ZTP インストール後は、このラベルを操作しないでください。

リンクされた CR

自動的に作成された **ClusterGroupUpgrade** CR には所有者の参照が、そこから派生した **ManagedCluster** として設定されます。この参照により、**ManagedCluster** CR を削除すると、**ClusterGroupUpgrade** のインスタンスがサポートされるリソースと共に削除されるようにします。

4.2. GITOPS ZTP を使用したマネージドクラスターのデプロイの概要

Red Hat Advanced Cluster Management (RHACM) は、GitOps Zero Touch Provisioning (ZTP) を使用して、単一ノードの OpenShift Container Platform クラスター、3 ノードのクラスター、および標準クラスターをデプロイします。サイト設定データは、Git リポジトリーで OpenShift Container Platform カスタムリソース (CR) として管理します。GitOps ZTP は、宣言的な GitOps アプローチを使用して、一度開発すればどこにでもデプロイできるモデルを使用して、マネージドクラスターをデプロイします。

クラスターのデプロイメントには、以下が含まれます。

- ホストオペレーティングシステム (RHCOS) の空のサーバーへのインストール。
- OpenShift Container Platform のデプロイ
- クラスターポリシーおよびサイトサブスクリプションの作成
- サーバーオペレーティングシステムに必要なネットワーク設定を行う
- プロファイル Operator をデプロイし、パフォーマンスプロファイル、PTP、SR-IOV などの必要なソフトウェア関連の設定を実行します。

マネージドサイトのインストールプロセスの概要
マネージドサイトのカスタムリソース (CR) をハブクラスターに適用すると、次のアクションが自動的に実行されます。

1. Discovery イメージの ISO ファイルが生成され、ターゲットホストで起動します。
2. ISO ファイルがターゲットホストで正常に起動すると、ホストのハードウェア情報が RHACM にレポートされます。
3. すべてのホストの検出後に、OpenShift Container Platform がインストールされます。
4. OpenShift Container Platform のインストールが完了すると、ハブは **klusterlet** サービスをターゲットクラスターにインストールします。
5. 要求されたアドオンサービスがターゲットクラスターにインストールされている。

マネージドクラスターの **Agent** CR がハブクラスター上に作成されると、検出イメージ ISO プロセスが完了します。

重要

ターゲットのベアメタルホストは、vDU アプリケーションワークロードに推奨される単一ノード OpenShift クラスター設定 に記載されているネットワーク、ファームウェア、およびハードウェアの要件を満たす必要があります。

4.3. マネージドベアメタルホストシークレットの作成

マネージドベアメタルホストに必要な **Secret** カスタムリソース (CR) をハブクラスターに追加します。GitOps Zero Touch Provisioning (ZTP) パイプラインが Baseboard Management Controller (BMC) にアクセスするためのシークレットと、アシストインストーラーサービスがレジストリーからクラスターインストールイメージを取得するためのシークレットが必要です。

注記

シークレットは、**SiteConfig** CR から名前で参照されます。namespace は **SiteConfig** namespace と一致する必要があります。

手順

1. ホスト Baseboard Management Controller (BMC) の認証情報と、OpenShift およびすべてのアドオンクラスター Operator のインストールに必要なプルシークレットを含む YAML シークレットファイルを作成します。

   a. 次の YAML をファイル **example-sno-secret.yaml** として保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno (3)
data:
  .dockerconfigjson: <pull_secret> (4)
type: kubernetes.io/dockerconfigjson
```

(1) 関連する **SiteConfig** CR で設定された namespace と一致する必要があります

(2) **password** と **username** の Base64 エンコード値

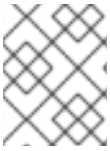
(3) 関連する **SiteConfig** CR で設定された namespace と一致する必要があります

(4) Base64 でエンコードされたプルシークレット

2. **example-sno-secret.yaml** への相対パスを、クラスターのインストールに使用する **kustomization.yaml** ファイルに追加します。

4.4. GITOPS ZTP を使用したインストール用の DISCOVERY ISO カーネル引数の設定

GitOps Zero Touch Provisioning (ZTP) ワークフローは、マネージドベアメタルホストでの OpenShift Container Platform インストールプロセスの一部として Discovery ISO を使用します。**InfraEnv** リソースを編集して、Discovery ISO のカーネル引数を指定できます。これは、特定の環境要件を持つクラスターのインストールに役立ちます。たとえば、Discovery ISO の **rd.net.timeout.carrier** カーネル引数を設定して、クラスターの静的ネットワーク設定を容易にしたり、インストール中に root ファイルシステムをダウンロードする前に DHCP アドレスを受信したりできます。



注記

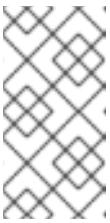
OpenShift Container Platform 4.15 では、カーネル引数の追加のみを行うことができます。カーネル引数を置き換えたり削除したりすることはできません。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. **InfraEnv** CR を作成し、**spec.kernelArguments** 仕様を編集してカーネル引数を設定します。
   a. 次の YAML を **InfraEnv-example.yaml** ファイルに保存します。



注記

この例の **InfraEnv** CR は、**SiteConfig** CR の値に基づいて入力される **{{ .Cluster.ClusterName }}** などのテンプレート構文を使用します。**SiteConfig** CR は、デプロイメント中にこれらのテンプレートの値を自動的に設定します。テンプレートを手動で編集しないでください。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "1"
  name: "{{ .Cluster.ClusterName }}"
  namespace: "{{ .Cluster.ClusterName }}"
spec:
  clusterRef:
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
  kernelArguments:
    - operation: append (1)
      value: audit=0 (2)
    - operation: append
      value: trace=1
```

```
    sshAuthorizedKey: "{{ .Site.SshPublicKey }}"
    proxy: "{{ .Cluster.ProxySettings }}"
    pullSecretRef:
      name: "{{ .Site.PullSecretRef.Name }}"
    ignitionConfigOverride: "{{ .Cluster.IgnitionConfigOverride }}"
    nmStateConfigLabelSelector:
      matchLabels:
        nmstate-label: "{{ .Cluster.ClusterName }}"
    additionalNTPSources: "{{ .Cluster.AdditionalNTPSources }}"
```

1. カーネル引数を追加するには、追加操作を指定します。
2. 設定するカーネル引数を指定します。この例では、audit カーネル引数と trace カーネル引数を設定します。

2. **InfraEnv-example.yaml** CR を、Git リポジトリー内の **SiteConfig** CR と同じ場所にコミットし、変更をプッシュします。次の例は、サンプルの Git リポジトリー構造を示しています。

```
~/example-ztp/install
        └── site-install
            ├── siteconfig-example.yaml
            ├── InfraEnv-example.yaml
            ...
```

3. **SiteConfig** CR の **spec.clusters.crTemplates** 仕様を編集して、Git リポジトリーの **InfraEnv-example.yaml** CR を参照します。

```
clusters:
  crTemplates:
    InfraEnv: "InfraEnv-example.yaml"
```

SiteConfig CR をコミットおよびプッシュしてクラスターをデプロイする準備ができたら、ビルドパイプラインは Git リポジトリー内のカスタム **InfraEnv-example** CR を使用して、カスタムカーネル引数を含むインフラストラクチャー環境を設定します。

検証

カーネル引数が適用されていることを確認するには、Discovery イメージが OpenShift Container Platform をインストールする準備ができていることを確認した後、インストールプロセスを開始する前にターゲットホストに SSH 接続します。その時点で、**/proc/cmdline** ファイルで Discovery ISO のカーネル引数を表示できます。

1. ターゲットホストとの SSH セッションを開始します。

```
$ ssh -i /path/to/privatekey core@<host_name>
```

2. 次のコマンドを使用して、システムのカーネル引数を表示します。

```
$ cat /proc/cmdline
```

4.5. SITECONFIG と GITOPS ZTP を使用したマネージドクラスターのデプロイ

次の手順を使用して、**SiteConfig** カスタムリソース (CR) と関連ファイルを作成し、GitOps Zero Touch Provisioning (ZTP) クラスターのデプロイメントを開始します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセスできる必要があり、ArgoCD アプリケーションのソースリポジトリーとして設定する必要があります。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。

注記

ソースリポジトリーを作成するときは、**ztp-site-generate** コンテナーから抽出した **argocd/deployment/argocd-openshift-gitops-patch.json** パッチファイルを使用して ArgoCD アプリケーションにパッチを適用してください。「ArgoCD を使用したハブクラスターの設定」を参照してください。

- マネージドクラスターをプロビジョニングする準備を整えるには、各ベアメタルホストごとに次のものが必要です。

  ネットワーク接続

  ネットワークには DNS が必要です。マネージドクラスターホストは、ハブクラスターから到達可能である必要があります。ハブクラスターとマネージドクラスターホストの間にレイヤー 3 接続が存在することを確認します。

  Baseboard Management Controller (BMC) の詳細

  GitOps ZTP は、BMC のユーザー名とパスワードの詳細を使用して、クラスターのインストール中に BMC に接続します。GitOps ZTP プラグインは、サイトの Git リポジトリーの **SiteConfig** CR に基づいて、ハブクラスター上の **ManagedCluster** CR を管理します。ホストごとに個別の **BMCSecret** CR を手動で作成します。

  手順

  1. ハブクラスターで必要なマネージドクラスターシークレットを作成します。これらのリソースは、クラスター名と一致する名前を持つネームスペースに存在する必要があります。たとえば、**out/argocd/example/siteconfig/example-sno.yaml** では、クラスター名と namespace が **example-sno** になっています。

     a. 次のコマンドを実行して、クラスター namespace をエクスポートします。

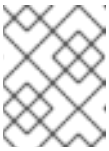
     ```
     $ export CLUSTERNS=example-sno
     ```

     b. namespace を作成します。

     ```
     $ oc create namespace $CLUSTERNS
     ```

  2. マネージドクラスターのプルシークレットと BMC **Secret** CR を作成します。プルシークレットには、OpenShift Container Platform のインストールに必要なすべての認証情報と、必要なすべての Operator を含める必要があります。詳細は、「マネージドベアメタルホス

トシークレットの作成」を参照してください。



注記

シークレットは、名前で **SiteConfig** カスタムリソース (CR) から参照されます。namespace は **SiteConfig** namespace と一致する必要があります。

3. Git リポジトリーのローカルクローンに、クラスターの **SiteConfig** CR を作成します。

   a. **out/argocd/example/siteconfig/** フォルダーから CR の適切な例を選択します。フォルダーには、単一ノード、3 ノード、標準クラスターのサンプルファイルが含まれます。

      - **example-sno.yaml**
      - **example-3node.yaml**
      - **example-standard.yaml**

   b. サンプルファイルのクラスターおよびホスト詳細を、必要なクラスタータイプに一致するように変更します。以下に例を示します。

シングルノード OpenShift SiteConfig CR の例

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created
under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    # installConfigOverrides is a generic way of passing install-config
    # parameters through the siteConfig.  The 'capabilities' field configures
    # the composable openshift feature.  In this 'capabilities' setting, we
    # remove all but the marketplace component from the optional set of
    # components.
    # Notes:
    # - OperatorLifecycleManager is needed for 4.15 and later
    # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
    # - Ingress is needed for 4.16 and later
    installConfigOverrides: |
      {
        "capabilities": {
          "baselineCapabilitySet": "None",
          "additionalEnabledCapabilities": [
            "NodeTuning",
            "OperatorLifecycleManager"
```

```
          "Ingress"
        ]
      }
    }
  # It is strongly recommended to include crun manifests as part of the additional
install-time manifests for 4.13+.
  # The crun manifests can be obtained from source-crs/optional-extra-manifest/
and added to the git repo ie.sno-extra-manifest.
  # extraManifestPath: sno-extra-manifest
  clusterLabels:
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples
    du-profile: "latest"
    # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples in ../policygentemplates:
    # ../policygentemplates/common-ranGen.yaml will apply to all clusters with
'common: true'
    common: true
    # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all clusters with
'group-du-sno: ""'
    group-du-sno: ""
    # ../policygentemplates/example-sno-site.yaml will apply to all clusters with 'sites:
"example-sno"'
    # Normally this should match or contain the cluster name so it only applies to a
single cluster
    sites : "example-sno"
  clusterNetwork:
    - cidr: 1001:1::/48
      hostPrefix: 64
  machineNetwork:
    - cidr: 1111:2222:3333:4444::/64
  serviceNetwork:
    - 1001:2::/112
  additionalNTPSources:
    - 1111:2222:3333:4444::2
  # Initiates the cluster for workload partitioning. Setting specific reserved/isolated
CPUSets is done via PolicyTemplate
  # please see Workload Partitioning Feature for a complete guide.
  cpuPartitioningMode: AllNodes
  # Optionally; This can be used to override the KlusterletAddonConfig that is
created for this cluster:
  #crTemplates:
  #  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
  nodes:
    - hostName: "example-node1.example.com"
      role: "master"
      # Optionally; This can be used to configure desired BIOS setting on a host:
      #biosConfigRef:
      #  filePath: "example-hw.profile"
      bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embed
ded.1"
      bmcCredentialsName:
        name: "example-node1-bmh-secret"
      bootMACAddress: "AA:BB:CC:DD:EE:11"
      # Use UEFISecureBoot to enable secure boot
```

```
      bootMode: "UEFI"
      rootDeviceHints:
        deviceName: "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0"
      # disk partition at `/var/lib/containers` with ignitionConfigOverride. Some values
must be updated. See DiskPartitionContainer.md for more details
      ignitionConfigOverride: |
        {
         "ignition": {
          "version": "3.2.0"
         },
         "storage": {
          "disks": [
           {
            "device": "/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62",
            "partitions": [
             {
              "label": "var-lib-containers",
              "sizeMiB": 0,
              "startMiB": 250000
             }
          ],
          "wipeTable": false
         }
        ],
         "filesystems": [
          {
           "device": "/dev/disk/by-partlabel/var-lib-containers",
           "format": "xfs",
           "mountOptions": [
            "defaults",
            "prjquota"
           ],
           "path": "/var/lib/containers",
           "wipeFilesystem": true
          }
         ]
        },
        "systemd": {
         "units": [
          {
           "contents": "# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-
disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-
disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/di
sk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
           "enabled": true,
           "name": "var-lib-containers.mount"
          }
         ]
        }
       }
      nodeNetwork:
        interfaces:
         - name: eno1
```

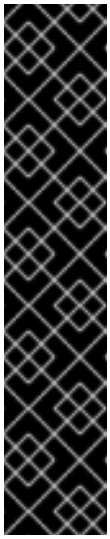
```
      macAddress: "AA:BB:CC:DD:EE:11"
  config:
    interfaces:
      - name: eno1
        type: ethernet
        state: up
        ipv4:
          enabled: false
        ipv6:
          enabled: true
          address:
          # For SNO sites with static IP addresses, the node-specific,
          # API and Ingress IPs should all be the same and configured on
          # the interface
          - ip: 1111:2222:3333:4444::aaaa:1
            prefix-length: 64
    dns-resolver:
      config:
        search:
        - example.com
        server:
        - 1111:2222:3333:4444::2
    routes:
      config:
      - destination: ::/0
        next-hop-interface: eno1
        next-hop-address: 1111:2222:3333:4444::1
        table-id: 254
```

注記

BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。読みやすくするために、**installConfigOverrides** および **ignitionConfigOverride** フィールドは例に拡張されています。

c. **out/argocd/extra-manifest** で extra-manifest **MachineConfig** CR のデフォルトセットを検査できます。これは、インストール時にクラスターに自動的に適用されます。

d. オプション: プロビジョニングされたクラスターに追加のインストール時マニフェストをプロビジョニングするには、Git リポジトリーに **sno-extra-manifest/** などのディレクトリーを作成し、このディレクトリーにカスタムマニフェストの CR を追加します。**SiteConfig.yaml** が **extraManifestPath** フィールドでこのディレクトリーを参照する場合、この参照ディレクトリーの CR はすべて、デフォルトの追加マニフェストセットに追加されます。



CRUN OCI コンテナーランタイムの有効化

クラスターのパフォーマンスを最適化するには、シングルノード OpenShift、追加のワーカーノードを備えたシングルノード OpenShift、3 ノード OpenShift、および標準クラスターのマスターノードとワーカーノードで crun を有効にします。

クラスターの再起動を回避するには、追加の Day 0 インストール時マニフェストとして **ContainerRuntimeConfig** CR で crun を有効にします。

enable-crun-master.yaml および **enable-crun-worker.yaml** CR ファイルは、**ztp-site-generate** コンテナーから抽出できる **out/source-crs/optional-extra-manifest/** フォルダーにあります。詳細は、「GitOps ZTP パイプラインでの追加インストールマニフェストのカスタマイズ」を参照してください。

4. **out/argocd/example/siteconfig/kustomization.yaml** に示す例のように、**generators** セクションの **kustomization.yaml** ファイルに **SiteConfig** CR を追加してください。

5. **SiteConfig** CR と関連する **kustomization.yaml** の変更を Git リポジトリーにコミットし、変更をプッシュします。
ArgoCD パイプラインが変更を検出し、マネージドクラスターのデプロイを開始します。

検証

- ノードのデプロイ後にカスタムのロールとラベルが適用されていることを確認します。

```
$ oc describe node example-node.example.com
```

出力例

```
Name:   example-node.example.com
Roles:  control-plane,example-label,master,worker
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        custom-label/parameter1=true
        kubernetes.io/arch=amd64
        kubernetes.io/hostname=cnfdf03.telco5gran.eng.rdu2.redhat.com
        kubernetes.io/os=linux
        node-role.kubernetes.io/control-plane=
        node-role.kubernetes.io/example-label= (1)
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

(1) カスタムラベルがノードに適用されます。

関連情報

- シングルノード OpenShift SiteConfig CR インストールリファレンス

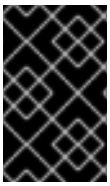
4.5.1. GitOps ZTP の高速プロビジョニング

重要

GitOps ZTP のアクセラレートプロビジョニングは、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 製品サポート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場合があります。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。

このリリースにより、シングルノード OpenShift の GitOps ZTP の高速プロビジョニングを使用することで、クラスターのインストールにかかる時間を短縮できます。高速 ZTP は、ポリシーから派生した Day 2 マニフェストを早い段階で適用することで、インストールを高速化します。



重要

GitOps ZTP の高速化は、アシステッドインストーラーを使用して単一ノードの OpenShift をインストールする場合にのみサポートされます。そうしないと、このインストール方法は失敗します。

4.5.1.1. 高速化された ZTP のアクティブ化

次の例のように、**spec.clusters.clusterLabels.accelerated-ztp** ラベルを使用して高速 ZTP をアクティブ化できます。

アクセラレート ZTP SiteConfig CR の例。

```
apiVersion: ran.openshift.io/v2
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  # ...
    clusterLabels:
      common: true
      group-du-sno: ""
      sites : "example-sno"
      accelerated-ztp: full
```

accelerated-ztp: full を使用して、高速化されたプロセスを完全に自動化できます。GitOps ZTP は、高速化された GitOps ZTP **ConfigMap** への参照を使用して **AgentClusterInstall** リソースを更新し、TALM によってポリシーから抽出されたリソースと、ZTP ジョブマニフェストを高速化します。

accelerated-ztp: partial を使用する場合、GitOps ZTP には高速化されたジョブマニフェストは含まれませんが、次の **種類** のクラスターのインストール時に作成されたポリシー派生オブジェクトが含まれます。

- **PerformanceProfile.performance.openshift.io**
- **Tuned.tuned.openshift.io**
- **namespace**
- **CatalogSource.operators.coreos.com**
- **ContainerRuntimeConfig.machineconfiguration.openshift.io**

この部分的な高速化により、**パフォーマンスプロファイル**、**Tuned**、および **ContainerRuntimeConfig** タイプのリソースを適用する際に、ノードが実行する再起動の数を減らすことができます。TALM は、RHACM がクラスターのインポートを完了した後、標準の GitOps ZTP と同じフローに従って、ポリシーから派生する Operator サブスクリプションをインストールします。

デプロイメントの規模により、迅速な ZTP の利点が向上します。**accelerated-ztp: full** を使用すると、多数のクラスターをより利点を得ることができます。クラスターの数が少ない場合、インストール時間の短縮はそれほど大きくありません。完全な高速化された ZTP は、namespace の背後に残り、手動で削除する必要のあるスポーク上で完了したジョブのままになります。

accelerated-ztp: partial を使用する利点の 1 つは、株式実装で問題が発生した場合に、またはカスタム機能が必要な場合に、オンスポークジョブの機能を上書きすることができることです。

4.5.1.2. 高速化された ZTP プロセス

アクセラレート ZTP は、追加の **ConfigMap** を使用して、スポーククラスターのポリシーから派生したリソースを作成します。標準の **ConfigMap** には、GitOps ZTP ワークフローがクラスターのインストールをカスタマイズするために使用するマニフェストが含まれています。

TALM は、**accelerated-ztp** ラベルが設定されていることを検出し、2 番目の **ConfigMap** を作成します。高速化された ZTP の一部として、**SiteConfig** ジェネレーターは、< **spoke-cluster-name>-aztp** 命名規則を使用して、2 番目の **ConfigMap** への参照を追加します。

TALM が 2 つ目の **ConfigMap** を作成した後、マネージドクラスターにバインドされたすべてのポリシーを見つけ、GitOps ZTP プロファイル情報を抽出します。TALM は GitOps ZTP プロファイル情報を < **spoke-cluster-name>-aztp ConfigMap** カスタムリソース(CR)に追加し、CR をハブクラスター API に適用します。

4.5.2. GitOps ZTP および SiteConfig リソースを使用した単一ノード OpenShift クラスターの IPsec 暗号化の設定

GitOps ZTP および Red Hat Advanced Cluster Management (RHACM)を使用してインストールする管理対象ノード OpenShift クラスターで IPsec 暗号化を有効にできます。マネージドクラスターの外部にある Pod と IPsec エンドポイント間の外部トラフィックを暗号化できます。OVN-Kubernetes クラスターネットワーク上のノード間のすべての Pod 間ネットワークトラフィックが、Transport モード の IPsec で暗号化されます。

注記

OpenShift Container Platform 4.16 では、GitOps ZTP および RHACM を使用して IPsec 暗号化をデプロイすることは、単一ノードの OpenShift クラスターに対してのみ検証されます。

GitOps ZTP IPsec 実装は、リソースに制約のあるプラットフォームにデプロイすることを前提としています。そのため、単一の **MachineConfig** CR を使用した機能のみをインストールし、前提条件として単一ノードの OpenShift クラスターに NMState Operator をインストールする必要はありません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- マネージドクラスターに必要なインストールおよびポリシーのカスタムリソース(CR)を生成するために、RHACM とハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。
- **butane** ユーティリティー（バージョン 0.20.0 以降）をインストールしている。
- IPsec エンドポイントの PKCS#12 証明書と PEM 形式の CA 証明書がある。

手順

1. **ztp-site-generate** コンテナーソースの最新バージョンを抽出し、カスタムサイト設定データを管理するリポジトリーにマージします。
2. クラスターで IPsec を設定する必要な値を使用して、**optional-extra-manifest/ipsec/ipsec-endpoint-config.yaml** を設定します。以下に例を示します。

```
interfaces:
- name: hosta_conn
  type: ipsec
  libreswan:
    left: <cluster_node> 1
    leftid: '%fromcert'
    leftmodecfgclient: false
    leftcert: <left_cert> 2
    leftrsasigkey: '%cert'
    right: <external_host> 3
    rightid: '%fromcert'
    rightrsasigkey: '%cert'
    rightsubnet: <external_address> 4
    ikev2: insist 5
    type: tunnel
```

1 & **lt;cluster_node&** gt; は、クラスター側の IPsec トンネルのクラスターノードの IP アドレスまたは DNS 名に置き換えます。

2 <**left_cert**> を IPsec 証明書のニックネームに置き換えます。

3 外部ホストの IP アドレスまたはホスト名または DNS 名に置き換え **<external_host>** ます。

4 IPsec トンネルの反対側の外部ホストの IP アドレスまたはサブネットに置き換え **<external_address>** ます。

5 IKEv2 VPN 暗号化プロトコルのみを使用してください。非推奨の IKEv1 は使用しないでください。

3. **ca.pem** および **left_server.p12** 証明書を **optional-extra-manifest/ipsec** フォルダーに追加します。証明書ファイルは、各ホストの Network Security Services (NSS)データベースに必要です。これらのファイルは、後の手順で Butane 設定の一部としてインポートされます。

   a. **left_server.p12**: IPsec エンドポイントの証明書バンドル

   b. **ca.pem**: 証明書に署名した認証局

4. Git リポジトリーの **optional-extra-manifest/ipsec** フォルダーでシェルプロンプトを開き、カスタムサイト設定データを維持します。

5. **optional-extra-manifest/ipsec/build.sh** スクリプトを実行して、必要な Butane および **MachineConfig** CR ファイルを生成します。

   出力例

```
out
└── argocd
    └── example
        └── optional-extra-manifest
            └── ipsec
                ├── 99-ipsec-master-endpoint-config.bu 1
                ├── 99-ipsec-master-endpoint-config.yaml
                ├── 99-ipsec-worker-endpoint-config.bu
                ├── 99-ipsec-worker-endpoint-config.yaml
                ├── build.sh
                ├── ca.pem 2
                ├── left_server.p12
                ├── enable-ipsec.yaml
                ├── ipsec-endpoint-config.yml
                └── README.md
```

1 **ipsec/build.sh** スクリプトは、Butane およびエンドポイント設定 CR を生成します。

2 ご自分のネットワークに関連する **ca.pem** および **left_server.p12** 証明書ファイルを指定します。

6. カスタムサイトの設定データを管理するリポジトリーに **custom-manifest/** フォルダーを作成します。**enable-ipsec.yaml** および **99-ipsec-*** YAML ファイルをディレクトリーに追加します。以下に例を示します。

```
siteconfig
  ├── site1-sno-du.yaml
```

```
├── extra-manifest/
└── custom-manifest
    ├── enable-ipsec.yaml
    ├── 99-ipsec-worker-endpoint-config.yaml
    └── 99-ipsec-master-endpoint-config.yaml
```

7. **SiteConfig** CR で、**custom-manifest/** ディレクトリーを **extraManifests.searchPaths** フィールドに追加します。以下に例を示します。

```
clusters:
- clusterName: "site1-sno-du"
  networkType: "OVNKubernetes"
  extraManifests:
    searchPaths:
      - extra-manifest/
      - custom-manifest/
```

8. **SiteConfig** CR の変更内容と更新されたファイルを Git リポジトリーにコミットし、変更をプッシュして、マネージドクラスターをプロビジョニングし、IPsec 暗号化を設定します。ArgoCD パイプラインが変更を検出し、マネージドクラスターのデプロイを開始します。

   クラスターのプロビジョニング中に、GitOps ZTP パイプラインは、/**custom-manifest** ディレクトリー内の CR を、**extra-manifest/** に保存されている追加マニフェストのデフォルトのセットに追加します。

検証

管理対象の単一ノードの OpenShift クラスターで IPsec 暗号化が正常に適用されたことを確認するには、以下の手順を実行します。

1. 次のコマンドを実行して、マネージドクラスターのデバッグ Pod を起動します。

```
$ oc debug node/<node_name>
```

2. IPsec ポリシーがクラスターノードに適用されていることを確認します。

```
sh-5.1# ip xfrm policy
```

出力例

```
src 172.16.123.0/24 dst 10.1.232.10/32
  dir out priority 1757377 ptype main
  tmpl src 10.1.28.190 dst 10.1.232.10
    proto esp reqid 16393 mode tunnel
src 10.1.232.10/32 dst 172.16.123.0/24
  dir fwd priority 1757377 ptype main
  tmpl src 10.1.232.10 dst 10.1.28.190
    proto esp reqid 16393 mode tunnel
src 10.1.232.10/32 dst 172.16.123.0/24
  dir in priority 1757377 ptype main
  tmpl src 10.1.232.10 dst 10.1.28.190
    proto esp reqid 16393 mode tunnel
```

3. IPsec トンネルが起動し、接続されていることを確認します。

```
sh-5.1# ip xfrm state
```

出力例

```
src 10.1.232.10 dst 10.1.28.190
  proto esp spi 0xa62a05aa reqid 16393 mode tunnel
  replay-window 0 flag af-unspec esn
  auth-trunc hmac(sha1) 0x8c59f680c8ea1e667b665d8424e2ab749cec12dc 96
  enc cbc(aes)
0x2818a489fe84929c8ab72907e9ce2f0eac6f16f2258bd22240f4087e0326badb
  anti-replay esn context:
   seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
   replay_window 128, bitmap-length 4
   00000000 00000000 00000000 00000000
src 10.1.28.190 dst 10.1.232.10
  proto esp spi 0x8e96e9f9 reqid 16393 mode tunnel
  replay-window 0 flag af-unspec esn
  auth-trunc hmac(sha1) 0xd960ddc0a6baaccb343396a51295e08cfd8aaddd 96
  enc cbc(aes)
0x0273c02e05b4216d5e652de3fc9b3528fea94648bc2b88fa01139fdf0beb27ab
  anti-replay esn context:
   seq-hi 0x0, seq 0x0, oseq-hi 0x0, oseq 0x0
   replay_window 128, bitmap-length 4
   00000000 00000000 00000000 00000000
```

4. 外部ホストサブネットの既知の IP に ping します。たとえば、**ipsec/ipsec-endpoint-config.yaml** で設定した **rightsubnet** 範囲で IP に ping します。

```
sh-5.1# ping 172.16.110.8
```

出力例

```
sh-5.1# ping 172.16.110.8
PING 172.16.110.8 (172.16.110.8) 56(84) bytes of data.
64 bytes from 172.16.110.8: icmp_seq=1 ttl=64 time=153 ms
64 bytes from 172.16.110.8: icmp_seq=2 ttl=64 time=155 ms
```

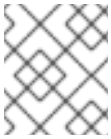
関連情報

- IPsec 暗号化の設定
- 暗号化プロトコルおよび IPsec モード
- RHACM および SiteConfig リソースを使用したマネージドクラスターのインストール

4.5.3. シングルノード OpenShift SiteConfig CR インストールリファレンス

表4.1 シングルノード OpenShift クラスター用の SiteConfig CR インストールオプション

SiteConfig CR フィールド	説明
spec.cpuPartitioningMode	**cpuPartitioningMode** の値を **AllNodes** に設定して、ワークロードパーティショニングを設定します。設定を完了するには、**PerformanceProfile** CR で **isolated** および **reserved** CPU を指定します。  **注記** **SiteConfig** CR の **cpuPartitioningMode** フィールドを使用したワークロードパーティショニングの設定は、OpenShift Container Platform 4.13 のテクノロジープレビュー機能です。
metadata.name	**name** を **Assisted-deployment-pull-secret** に設定し、**SiteConfig** CR と同じ namespace に **assisted-deployment-pull-secret** CR を作成します。
spec.clusterImageSetNameRef	サイト内のすべてのクラスターのハブクラスターで使用できるイメージセットを設定します。ハブクラスターでサポートされるバージョンの一覧を表示するには、**oc get clusterimagesets** を実行します。
installConfigOverrides	クラスターのインストール前に、**installConfigOverrides** フィールドを設定して、オプションのコンポーネントを有効または無効にします。  **重要** **SiteConfig** CR の例で指定されている参照設定を使用します。追加のコンポーネントをシステムに再度追加するには、追加の予約済み CPU 容量が必要になる場合があります。
spec.clusters.clusterImageSetNameRef	個々のクラスターをデプロイするために使用されるクラスターイメージセットを指定します。定義されている場合、サイトレベルで **spec.clusterImageSetNameRef** をオーバーライドします。
spec.clusters.clusterLabels	定義した **PolicyGenerator** または **PolicyGentemplate** CR のバインディングルールに対応するようにクラスターラベルを設定します。**PolicyGenerator** CR は **policyDefaults.placement.labelSelector** フィールドを使用します。**PolicyGenTemplate CR は spec.bindingRules** フィールドを使用します。 たとえば、**acmpolicygenerator/acm-common-ranGen.yaml** は **common: true** が設定されたすべてのクラスターに適用され、**acmpolicygenerator/acm-group-du-sno-ranGen.yaml** は **group-du-sno: ""** が設定されたすべてのクラスターに適用されます。
spec.clusters.crTemplates.KlusterletAddonConfig	オプション: **klusterletaddonconfig** を、クラスター用に作成された **KlusterletAddonConfigOverride.yaml to override the default `KlusterletAddonConfig** に設定します。

SiteConfig CR フィールド	説明
spec.clusters.nodes.hostName	単一ノードの導入では、単一のホストを定義します。3 ノードのデプロイメントの場合、3 台のホストを定義します。標準のデプロイメントでは、**role: master** と、**role: worker** で定義される 2 つ以上のホストを持つ 3 つのホストを定義します。
spec.clusters.nodes.nodeLabels	マネージドクラスター内のノードのカスタムロールを指定します。これらは追加のロールであり、OpenShift Container Platform コンポーネントでは使用されず、ユーザーによってのみ使用されます。カスタムロールを追加すると、そのロールの特定の設定を参照するカスタムマシン設定プールに関連付けることができます。インストール中にカスタムラベルまたはロールを追加すると、デプロイメントプロセスがより効率的になり、インストール完了後に追加の再起動が必要なくなります。
spec.clusters.nodes.automatedCleaningMode	オプション: コメントを解除して値を **metadata** に設定すると、ディスクを完全にワイプせずに、ディスクのパーティションテーブルのみを削除できるようになります。デフォルト値は、**disabled** です。
spec.clusters.nodes.bmcAddress	ホストへのアクセスに使用する BMC アドレス。すべてのクラスタータイプに適用されます。GitOps ZTP は、Redfish または IPMI プロトコルを使用して iPXE および仮想メディアの起動をサポートします。iPXE ブートを使用するには、RHACM 2.8 以降を使用する必要があります。BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。
spec.clusters.nodes.bmcAddress	ホストへのアクセスに使用する BMC アドレス。すべてのクラスタータイプに適用されます。GitOps ZTP は、Redfish または IPMI プロトコルを使用して iPXE および仮想メディアの起動をサポートします。iPXE ブートを使用するには、RHACM 2.8 以降を使用する必要があります。BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。  **注記** ファーエッジ通信会社のユースケースでは、GitOps ZTP では仮想メディアの使用のみがサポートされます。
spec.clusters.nodes.bmcCredentialsName	ホスト BMC 認証情報を使用して、別途作成した **bmh-secret** CR を設定します。**bmh-secret** CR を作成するときは、ホストをプロビジョニングする **SiteConfig** CR と同じ namespace を使用します。
spec.clusters.nodes.bootMode	ホストのブートモードを **UEFI** に設定します。デフォルト値は **UEFI** です。**UEFISecureBoot** を使用して、ホストでセキュアブートを有効にします。
spec.clusters.nodes.rootDeviceHints	導入するデバイスを指定します。再起動後も安定した識別子が推奨されます。例： **wwn: <disk_wwn** > または **deviceName: /dev/disk/by-path/<device_path&gt**;& **lt;by-path>** の値が優先されます。安定した識別子の詳細なリストは、「ルートデバイスのヒント」セクションを参照してください。

SiteConfig CR フィールド	説明
spec.clusters.nodes.ignitionConfigOverride	オプション: このフィールドを使用して、永続ストレージのパーティションを割り当てます。ディスク ID とサイズを特定のハードウェアに合わせて調整します。
spec.clusters.nodes.nodeNetwork	ノードのネットワーク設定を行います。
spec.clusters.nodes.nodeNetwork.config.interfaces.ipv6	ホストの IPv6 アドレスを設定します。静的 IP アドレスを持つ単一ノードの OpenShift クラスターの場合、ノード固有の API と Ingress IP は同じである必要があります。

関連情報

- GitOps ZTP パイプラインでの追加インストールマニフェストのカスタマイズ
- GitOps ZTP サイト設定リポジトリーの準備
- ArgoCD を使用したハブクラスターの設定
- バリデーターインフォームポリシーを使用した GitOps ZTP クラスターデプロイメントの完了のシグナリング
- マネージドベアメタルホストシークレットの作成
- BMC アドレス指定
- ルートデバイスヒントについて

4.6. マネージドクラスターのインストールの進行状況の監視

ArgoCD パイプラインは、**SiteConfig** CR を使用してクラスター設定 CR を生成し、それをハブクラスターと同期します。ArgoCD ダッシュボードでこの同期の進捗をモニターできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

同期が完了すると、インストールは一般的に以下のように行われます。

1. Assisted Service Operator は OpenShift Container Platform をクラスターにインストールします。次のコマンドを実行して、RHACM ダッシュボードまたはコマンドラインからクラスターのインストールの進行状況を監視できます。

   a. クラスター名をエクスポートします。

   ```
   $ export CLUSTER=<clusterName>
   ```

b. マネージドクラスターの **AgentClusterInstall** CR をクエリーします。

```
$ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Completed")]}' | jq
```

c. クラスターのインストールイベントを取得します。

```
$ curl -sk $(oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.debugInfo.eventsURL}')  | jq '.[-2,-1]'
```

4.7. インストール CR の検証による GITOPS ZTP のトラブルシューティング

ArgoCD パイプラインは **SiteConfig** と **PolicyGenTemplate** カスタムリソース (CR) を使用して、クラスター設定 CR と Red Hat Advanced Cluster Management (RHACM) ポリシーを生成します。以下の手順に従って、このプロセス時に発生する可能性のある問題のトラブルシューティングを行います。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. インストール CR が作成されたことは、以下のコマンドで確認することができます。

```
$ oc get AgentClusterInstall -n <cluster_name>
```

オブジェクトが返されない場合は、以下の手順を使用して ArgoCD パイプラインフローを **SiteConfig** ファイルからインストール CR にトラブルシューティングします。

2. ハブクラスターで **SiteConfig** CR を使用して **ManagedCluster** CR が生成されたことを確認します。

```
$ oc get managedcluster
```

3. **ManagedCluster** が見つからない場合は、**clusters** アプリケーションが Git リポジトリーからハブクラスターへのファイルの同期に失敗したかどうかを確認します。

```
$ oc describe -n openshift-gitops application clusters
```

a. **Status.Conditions** フィールドを確認して、マネージドクラスターのエラーログを表示します。たとえば、**SiteConfig** CR で **extraManifestPath:** に無効な値を設定すると、次のエラーが発生します。

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build /tmp/https___git.com/ran-sites/siteconfigs/ --enable-alpha-plugins` failed exit status 1: 2021/11/26 17:21:40 Error could not create extra-manifest ranSite1.extra-manifest3 stat extra-manifest3: no such file or directory 2021/11/26 17:21:40 Error: could not build the
```

```
entire SiteConfig defined by /tmp/kust-plugin-config-913473579: stat extra-manifest3: no
such file or directory Error: failure in plugin configured via /tmp/kust-plugin-config-
913473579; exit status 1: exit status 1
    Type:  ComparisonError
```

b. **Status.Sync** フィールドを確認します。ログエラーがある場合、**Status.Sync** フィールドは **Unknown** エラーを示している可能性があります。

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  clusters-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:             sites-config
        Repo URL:         https://git.com/ran-sites/siteconfigs/.git
        Target Revision:  master
    Status:               Unknown
```

4.8. SUPERMICRO サーバー上で起動する GITOPS ZTP 仮想メディアのトラブルシューティング

SuperMicro X11 サーバーは、イメージが **https** プロトコルを使用して提供される場合、仮想メディアのインストールをサポートしません。そのため、この環境のシングルノード OpenShift デプロイメントはターゲットノードで起動できません。この問題を回避するには、ハブクラスターにログインし、**Provisioning** リソースで Transport Layer Security (TLS) を無効にします。これにより、イメージアドレスで **https** スキームを使用している場合でも、イメージは TLS で提供されなくなります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. 次のコマンドを実行して、**Provisioning** リソースの TLS を無効にします。

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec":
{"disableVirtualMediaTLS": true}}'
```

2. シングルノード OpenShift クラスターをデプロイする手順を続行します。

4.9. GITOPS ZTP パイプラインからのマネージドクラスターサイトの削除

GitOps Zero Touch Provisioning (ZTP) パイプラインから、マネージドサイトと、関連するインストールおよび設定ポリシー CR を削除できます。

前提条件

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

- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. 関連する **SiteConfig** ファイルと **PolicyGenTemplate** ファイルを **kustomization.yaml** ファイルから削除して、サイトと関連する CR を削除します。
   GitOps ZTP パイプラインを再度実行すると、生成された CR は削除されます。

2. 任意: サイトを永続的に削除する場合は、Git リポジトリーから **SiteConfig** ファイルおよびサイト固有の **PolicyGenTemplate** ファイルも削除する必要があります。

3. 任意: たとえば、サイトを再デプロイする際にサイトを一時的に削除する場合には、Git リポジトリーに **SiteConfig** およびサイト固有の **PolicyGenTemplate** CR を残しておくことができます。

関連情報

- クラスターの削除について、詳しくは 管理からクラスターを削除する を参照してください。

4.10. GITOPS ZTP パイプラインからの古いコンテンツの削除

ポリシーの名前を変更した場合など、**PolicyGenTemplate** 設定を変更した結果、古いポリシーが作成された場合は、次の手順を使用して古いポリシーを削除します。

前提条件

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

- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. 影響を受ける **PolicyGenerator** または **PolicyGentemplate** ファイルを Git リポジトリーから削除し、コミットしてリモートリポジトリーにプッシュします。

2. アプリケーションを介して変更が同期され、影響を受けるポリシーがハブクラスターから削除されるのを待ちます。

3. 更新された **PolicyGenerator** または **PolicyGentemplate** ファイルを Git リポジトリーに追加し、リモートリポジトリーにコミットしてプッシュします。

注記

Git リポジトリーから GitOps Zero Touch Provisioning (ZTP) ポリシーを削除し、その結果としてハブクラスターからもポリシーが削除されても、マネージドクラスターの設定には影響しません。ポリシーとそのポリシーによって管理される CR は、マネージドクラスターに残ります。

4. 任意: 別の方法として、**PolicyGenTemplate** CR に変更を加えて古いポリシーを作成した後、これらのポリシーをハブクラスターから手動で削除することができます。ポリシーの削除は、RHACM コンソールから **Governance** タブを使用するか、以下のコマンドを使用して行うことができます。

```
$ oc delete policy -n <namespace> <policy_name>
```

4.11. GITOPS ZTP パイプラインの破棄

ArgoCD パイプラインと生成されたすべての GitOps Zero Touch Provisioning (ZTP) アーティファクトを削除できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

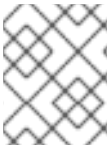
1. ハブクラスターの Red Hat Advanced Cluster Management (RHACM) からすべてのクラスターを切り離します。
2. 次のコマンドを使用して、**deployment** ディレクトリーの **kustomization.yaml** ファイルを削除します。

```
$ oc delete -k out/argocd/deployment
```

3. 変更をコミットして、サイトリポジトリーにプッシュします。

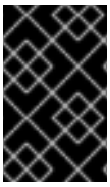
第5章 GITOPS ZTP を使用した単一ノードの OPENSHIFT クラスターの手動インストール

Red Hat Advanced Cluster Management (RHACM) とアシストサービスを使用して、管理対象の単一ノード OpenShift クラスターをデプロイできます。



注記

複数のマネージドクラスターを作成する場合は、ZTP を使用したファーエッジサイトのデプロイメント で説明されている **SiteConfig** メソッドを使用します。



重要

ターゲットのベアメタルホストは、vDU アプリケーションワークロードの推奨クラスター設定 に記載されているネットワーク、ファームウェア、およびハードウェアの要件を満たす必要があります。

5.1. GITOPS ZTP インストール CR と設定 CR の手動生成

ztp-site-generate コンテナーの **generator** エントリーポイントを使用して、**SiteConfig** および **PolicyGenTemplate** CR に基づいてクラスターのサイトインストールおよび設定カスタムリソース (CR) を生成します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. 次のコマンドを実行して、出力フォルダーを作成します。

```
$ mkdir -p ./out
```

2. **ztp-site-generate** コンテナーイメージから **argocd** ディレクトリーをエクスポートします。

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 extract /home/ztp --tar | tar x -C ./out
```

./out ディレクトリーの **out/argocd/example/** フォルダーには、参照 **PolicyGenTemplate** CR および **SiteConfig** CR があります。

出力例

```
out
└── argocd
    └── example
        ├── acmpolicygenerator
        │   ├── {policy-prefix}common-ranGen.yaml
        │   ├── {policy-prefix}example-sno-site.yaml
        │   ├── {policy-prefix}group-du-sno-ranGen.yaml
        │   ├── {policy-prefix}group-du-sno-validator-ranGen.yaml
```

```
│   ├── ...
│   ├── kustomization.yaml
│   └── ns.yaml
└── siteconfig
    ├── example-sno.yaml
    ├── KlusterletAddonConfigOverride.yaml
    └── kustomization.yaml
```

3. サイトインストール CR の出力フォルダーを作成します。

```
$ mkdir -p ./site-install
```

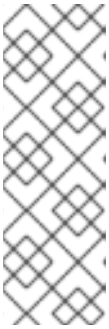
4. インストールするクラスタータイプのサンプル **SiteConfig** CR を変更します。**example-sno.yaml** を **site-1-sno.yaml** にコピーし、インストールするサイトとベアメタルホストの詳細に一致するように CR を変更します。次に例を示します。

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created under
same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    # installConfigOverrides is a generic way of passing install-config
    # parameters through the siteConfig.  The 'capabilities' field configures
    # the composable openshift feature.  In this 'capabilities' setting, we
    # remove all but the marketplace component from the optional set of
    # components.
    # Notes:
    # - OperatorLifecycleManager is needed for 4.15 and later
    # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
    # - Ingress is needed for 4.16 and later
    installConfigOverrides: |
      {
        "capabilities": {
          "baselineCapabilitySet": "None",
          "additionalEnabledCapabilities": [
            "NodeTuning",
            "OperatorLifecycleManager"
            "Ingress"
          ]
        }
      }
    # It is strongly recommended to include crun manifests as part of the additional install-time
manifests for 4.13+.
```

```
    # The crun manifests can be obtained from source-crs/optional-extra-manifest/ and added
to the git repo ie.sno-extra-manifest.
    # extraManifestPath: sno-extra-manifest
    clusterLabels:
      # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples
      du-profile: "latest"
      # These example cluster labels correspond to the bindingRules in the
PolicyGenTemplate examples in ../policygentemplates:
      # ../policygentemplates/common-ranGen.yaml will apply to all clusters with 'common: true'
      common: true
      # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all clusters with 'group-
du-sno: ""'
      group-du-sno: ""
      # ../policygentemplates/example-sno-site.yaml will apply to all clusters with 'sites:
"example-sno"'
      # Normally this should match or contain the cluster name so it only applies to a single
cluster
      sites : "example-sno"
    clusterNetwork:
      - cidr: 1001:1::/48
        hostPrefix: 64
    machineNetwork:
      - cidr: 1111:2222:3333:4444::/64
    serviceNetwork:
      - 1001:2::/112
    additionalNTPSources:
      - 1111:2222:3333:4444::2
    # Initiates the cluster for workload partitioning. Setting specific reserved/isolated CPUSets
is done via PolicyTemplate
    # please see Workload Partitioning Feature for a complete guide.
    cpuPartitioningMode: AllNodes
    # Optionally; This can be used to override the KlusterletAddonConfig that is created for this
cluster:
    #crTemplates:
    #  KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
    nodes:
      - hostName: "example-node1.example.com"
        role: "master"
        # Optionally; This can be used to configure desired BIOS setting on a host:
        #biosConfigRef:
        #  filePath: "example-hw.profile"
        bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embedded.1"

        bmcCredentialsName:
          name: "example-node1-bmh-secret"
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        # Use UEFISecureBoot to enable secure boot
        bootMode: "UEFI"
        rootDeviceHints:
          deviceName: "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0"
        # disk partition at `/var/lib/containers` with ignitionConfigOverride. Some values must be
updated. See DiskPartitionContainer.md for more details
        ignitionConfigOverride: |
          {
```

```
            "ignition": {
              "version": "3.2.0"
            },
            "storage": {
              "disks": [
                {
                  "device": "/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62",
                  "partitions": [
                    {
                      "label": "var-lib-containers",
                      "sizeMiB": 0,
                      "startMiB": 250000
                    }
                ],
                "wipeTable": false
              }
            ],
            "filesystems": [
              {
                "device": "/dev/disk/by-partlabel/var-lib-containers",
                "format": "xfs",
                "mountOptions": [
                  "defaults",
                  "prjquota"
                ],
                "path": "/var/lib/containers",
                "wipeFilesystem": true
              }
            ]
          },
          "systemd": {
            "units": [
              {
                "contents": "# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-
by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-
by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-
partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target",
                "enabled": true,
                "name": "var-lib-containers.mount"
              }
            ]
          }
        }
      nodeNetwork:
        interfaces:
          - name: eno1
            macAddress: "AA:BB:CC:DD:EE:11"
        config:
          interfaces:
            - name: eno1
              type: ethernet
              state: up
              ipv4:
                enabled: false
```

```
      ipv6:
        enabled: true
        address:
        # For SNO sites with static IP addresses, the node-specific,
        # API and Ingress IPs should all be the same and configured on
        # the interface
        - ip: 1111:2222:3333:4444::aaaa:1
          prefix-length: 64
    dns-resolver:
      config:
        search:
        - example.com
        server:
        - 1111:2222:3333:4444::2
    routes:
      config:
      - destination: ::/0
        next-hop-interface: eno1
        next-hop-address: 1111:2222:3333:4444::1
        table-id: 254
```



注記

ztp-site-generate コンテナーの **out/extra-manifest** ディレクトリーから参照 CR 設定ファイルを抽出したら、**extraManifests.searchPaths** を使用して、それらのファイルを含む git ディレクトリーへのパスを含めることができます。これにより、GitOps ZTP パイプラインはクラスターのインストール中にこれらの CR ファイルを適用できるようになります。**searchPaths** ディレクトリーを設定すると、GitOps ZTP パイプラインは、サイトのインストール中に **ztp-site-generate** コンテナーからマニフェストを取得しません。

5. 次のコマンドを実行して、変更された **SiteConfig** CR **site-1-sno.yaml** を処理し、Day 0 インストール CR を生成します。

```
$ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v `pwd`/site-install:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 generator install site-1-sno.yaml /output
```

出力例

```
site-install
└── site-1-sno
    ├── site-1_agentclusterinstall_example-sno.yaml
    ├── site-1-sno_baremetalhost_example-node1.example.com.yaml
    ├── site-1-sno_clusterdeployment_example-sno.yaml
    ├── site-1-sno_configmap_example-sno.yaml
    ├── site-1-sno_infraenv_example-sno.yaml
    ├── site-1-sno_klusterletaddonconfig_example-sno.yaml
    ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
    ├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
    ├── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
    ├── site-1-sno_managedcluster_example-sno.yaml
    ├── site-1-sno_namespace_example-sno.yaml
    └── site-1-sno_nmstateconfig_example-node1.example.com.yaml
```

6. オプション: **-E** オプションを使用して参照 **SiteConfig** CR を処理することにより、特定のクラスタータイプの Day 0 **MachineConfig** インストール CR のみを生成します。たとえば、以下のコマンドを実行します。

   a. **MachineConfig** CR の出力フォルダーを作成します。

   ```
   $ mkdir -p ./site-machineconfig
   ```

   b. **MachineConfig** インストール CR を生成します。

   ```
   $ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v `pwd`/site-machineconfig:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 generator install -E site-1-sno.yaml /output
   ```

   出力例

   ```
   site-machineconfig
   └── site-1-sno
       ├── site-1-sno_machineconfig_02-master-workload-partitioning.yaml
       ├── site-1-sno_machineconfig_predefined-extra-manifests-master.yaml
       └── site-1-sno_machineconfig_predefined-extra-manifests-worker.yaml
   ```

7. 前のステップの参照 **PolicyGenTemplate** CR を使用して、Day 2 の設定 CR を生成してエクスポートします。以下のコマンドを実行します。

   a. Day 2 CR の出力フォルダーを作成します。

   ```
   $ mkdir -p ./ref
   ```

   b. Day 2 設定 CR を生成してエクスポートします。

   ```
   $ podman run -it --rm -v `pwd`/out/argocd/example/acmpolicygenerator:/resources:Z -v `pwd`/ref:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16 generator config -N . /output
   ```

   このコマンドは、単一ノード OpenShift、3 ノードクラスター、および標準クラスター用のサンプルグループおよびサイト固有の **PolicyGenTemplate** CR を **./ref** フォルダーに生成します。

   出力例

   ```
   ref
   └── customResource
       ├── common
       ├── example-multinode-site
       ├── example-sno
       ├── group-du-3node
       ├── group-du-3node-validator
       │   └── Multiple-validatorCRs
       ├── group-du-sno
       ├── group-du-sno-validator
       ├── group-du-standard
       └── group-du-standard-validator
           └── Multiple-validatorCRs
   ```

8. クラスターのインストールに使用する CR のベースとして、生成された CR を使用します。「単一のマネージドクラスターのインストール」で説明されているように、インストール CR をハブクラスターに適用します。設定 CR は、クラスターのインストールが完了した後にクラスターに適用できます。

検証

- ノードのデプロイ後にカスタムのロールとラベルが適用されていることを確認します。

```
$ oc describe node example-node.example.com
```

出力例

```
Name:   example-node.example.com
Roles:  control-plane,example-label,master,worker
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        custom-label/parameter1=true
        kubernetes.io/arch=amd64
        kubernetes.io/hostname=cnfdf03.telco5gran.eng.rdu2.redhat.com
        kubernetes.io/os=linux
        node-role.kubernetes.io/control-plane=
        node-role.kubernetes.io/example-label= (1)
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

(1) カスタムラベルがノードに適用されます。

関連情報

- ワークロードの分割
- BMC アドレス指定
- ルートデバイスヒントについて
- シングルノード OpenShift SiteConfig CR インストールリファレンス

5.2. マネージドベアメタルホストシークレットの作成

マネージドベアメタルホストに必要な **Secret** カスタムリソース (CR) をハブクラスターに追加します。GitOps Zero Touch Provisioning (ZTP) パイプラインが Baseboard Management Controller (BMC) にアクセスするためのシークレットと、アシストインストーラーサービスがレジストリーからクラスターインストールイメージを取得するためのシークレットが必要です。

注記

シークレットは、**SiteConfig** CR から名前で参照されます。namespace は **SiteConfig** namespace と一致する必要があります。

手順

1. ホスト Baseboard Management Controller (BMC) の認証情報と、OpenShift およびすべてのアドオンクラスター Operator のインストールに必要なプルシークレットを含む YAML シークレットファイルを作成します。

   a. 次の YAML をファイル **example-sno-secret.yaml** として保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno (3)
data:
  .dockerconfigjson: <pull_secret> (4)
type: kubernetes.io/dockerconfigjson
```

   (1) 関連する **SiteConfig** CR で設定された namespace と一致する必要があります

   (2) **password** と **username** の Base64 エンコード値

   (3) 関連する **SiteConfig** CR で設定された namespace と一致する必要があります

   (4) Base64 でエンコードされたプルシークレット

2. **example-sno-secret.yaml** への相対パスを、クラスターのインストールに使用する **kustomization.yaml** ファイルに追加します。

5.3. GITOPS ZTP を使用した手動インストール用の DISCOVERY ISO カーネル引数の設定

GitOps Zero Touch Provisioning (ZTP) ワークフローは、マネージドベアメタルホストでの OpenShift Container Platform インストールプロセスの一部として Discovery ISO を使用します。**InfraEnv** リソースを編集して、Discovery ISO のカーネル引数を指定できます。これは、特定の環境要件を持つクラスターのインストールに役立ちます。たとえば、Discovery ISO の **rd.net.timeout.carrier** カーネル引数を設定して、クラスターの静的ネットワーク設定を容易にしたり、インストール中に root ファイルシステムをダウンロードする前に DHCP アドレスを受信したりできます。

注記

OpenShift Container Platform 4.15 では、カーネル引数の追加のみを行うことができます。カーネル引数を置き換えたり削除したりすることはできません。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- インストールと設定カスタムリソース (CR) を手動で生成している。

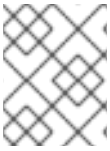
手順

1. **InfraEnv** CR の **spec.kernelArguments** 仕様を編集して、カーネル引数を設定します。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  kernelArguments:
    - operation: append (1)
      value: audit=0 (2)
    - operation: append
      value: trace=1
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  pullSecretRef:
    name: pull-secret
```

(1) カーネル引数を追加するには、追加操作を指定します。

(2) 設定するカーネル引数を指定します。この例では、audit カーネル引数と trace カーネル引数を設定します。



注記

SiteConfig CR は、Day-0 インストール CR の一部として **InfraEnv** リソースを生成します。

検証

カーネル引数が適用されていることを確認するには、Discovery イメージが OpenShift Container Platform をインストールする準備ができていることを確認した後、インストールプロセスを開始する前にターゲットホストに SSH 接続します。その時点で、**/proc/cmdline** ファイルで Discovery ISO のカーネル引数を表示できます。

1. ターゲットホストとの SSH セッションを開始します。

```
$ ssh -i /path/to/privatekey core@<host_name>
```

2. 次のコマンドを使用して、システムのカーネル引数を表示します。

```
$ cat /proc/cmdline
```

5.4. 単一のマネージドクラスターのインストール

アシストサービスと Red Hat Advanced Cluster Management (RHACM) を使用して、単一のマネージドクラスターを手動でデプロイできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- ベースボード管理コントローラー (BMC) **Secret** とイメージプルシークレット **Secret** カスタムリソース (CR) を作成しました。詳細は、「管理されたベアメタルホストシークレットの作成」を参照してください。
- ターゲットのベアメタルホストが、マネージドクラスターのネットワークとハードウェアの要件を満たしている。

手順

1. デプロイする特定のクラスターバージョンごとに **ClusterImageSet** を作成します (例: **clusterImageSet-4.15.yaml**)。**ClusterImageSet** のフォーマットは以下のとおりです。

```
apiVersion: hive.openshift.io/v1
kind: ClusterImageSet
metadata:
  name: openshift-4.16.0 (1)
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.16.0-x86_64 (2)
```

(1) デプロイする記述バージョン。

(2) デプロイする **releaseImage** を指定し、オペレーティングシステムイメージのバージョンを決定します。検出 ISO は、**releaseImage** で設定されたイメージバージョン、または正確なバージョンが利用できない場合は最新バージョンに基づいています。

2. **clusterImageSet** CR を適用します。

```
$ oc apply -f clusterImageSet-4.16.yaml
```

3. **cluster-namespace.yaml** ファイルに **Namespace** CR を作成します。

```
apiVersion: v1
kind: Namespace
metadata:
    name: <cluster_name> (1)
    labels:
      name: <cluster_name> (2)
```

(1) (2) プロビジョニングするマネージドクラスターの名前。

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

```
$ oc apply -f cluster-namespace.yaml
```

5. **ztp-site-generate** コンテナーから抽出し、要件を満たすようにカスタマイズした、生成された day-0 CR を適用します。

```
$ oc apply -R ./site-install/site-sno-1
```

関連情報

- マネージドクラスターネットワークの接続の前提条件
- シングルノード OpenShift クラスターへの LVM ストレージのデプロイ
- PolicyGenerator CR を使用した LVM ストレージの設定

5.5. マネージドクラスターのインストールステータスの監視

クラスターのステータスをチェックして、クラスターのプロビジョニングが正常に行われたことを確認します。

前提条件

- すべてのカスタムリソースが設定およびプロビジョニングされ、プロビジョニングされ、マネージドクラスターのハブで **Agent** カスタムリソースが作成されます。

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

True はマネージドクラスターの準備が整っていることを示します。

2. エージェントのステータスを確認します。

```
$ oc get agent -n <cluster_name>
```

3. **describe** コマンドを使用して、エージェントの条件に関する詳細な説明を指定します。認識できるステータスには、**BackendError**、**InputError**、**ValidationsFailing**、**InstallationFailed**、および **AgentIsConnected** が含まれます。これらのステータスは、**Agent** および **AgentClusterInstall** カスタムリソースに関連します。

```
$ oc describe agent -n <cluster_name>
```

4. クラスターのプロビジョニングのステータスを確認します。

```
$ oc get agentclusterinstall -n <cluster_name>
```

5. **describe** コマンドを使用して、クラスターのプロビジョニングステータスの詳細な説明を指定します。

```
$ oc describe agentclusterinstall -n <cluster_name>
```

6. マネージドクラスターのアドオンサービスのステータスを確認します。

```
$ oc get managedclusteraddon -n <cluster_name>
```

7. マネージドクラスターの **kubeconfig** ファイルの認証情報を取得します。

```
$ oc get secret -n <cluster_name> <cluster_name>-admin-kubeconfig -o jsonpath=
{.data.kubeconfig} | base64 -d > <directory>/<cluster_name>-kubeconfig
```

5.6. マネージドクラスターのトラブルシューティング

以下の手順を使用して、マネージドクラスターで発生する可能性のあるインストール問題を診断します。

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

出力例

```
NAME          HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE
AGE
SNO-cluster   true                                  True     True     2d19h
```

AVAILABLE 列のステータスが **True** の場合、マネージドクラスターはハブによって管理されます。

AVAILABLE 列のステータスが **Unknown** の場合、マネージドクラスターはハブによって管理されていません。その他の情報を取得するには、以下の手順を使用します。

2. **AgentClusterInstall** インストールのステータスを確認します。

```
$ oc get clusterdeployment -n <cluster_name>
```

出力例

```
NAME      PLATFORM          REGION   CLUSTERTYPE   INSTALLED   INFRAID
VERSION  POWERSTATE AGE
Sno0026   agent-baremetal                 false                 Initialized
2d14h
```

INSTALLED 列のステータスが **false** の場合、インストールは失敗していました。

3. インストールが失敗した場合は、以下のコマンドを実行して **AgentClusterInstall** リソースのステータスを確認します。

```
$ oc describe agentclusterinstall -n <cluster_name> <cluster_name>
```

4. エラーを解決し、クラスターをリセットします。

a. クラスターのマネージドクラスターリソースを削除します。

```
$ oc delete managedcluster <cluster_name>
```

b. クラスターの namespace を削除します。

```
$ oc delete namespace <cluster_name>
```

これにより、このクラスター用に作成された namespace スコープのカスタムリソースがすべて削除されます。続行する前に、**ManagedCluster** CR の削除が完了するのを待つ必要があります。

c. マネージドクラスターのカスタムリソースを再作成します。

5.7. RHACM によって生成されたクラスターインストール CR リファレンス

Red Hat Advanced Cluster Management (RHACM) は、サイトごとに **SiteConfig** CR を使用して生成する特定のインストールカスタムリソース (CR) のセットを使用して、単一ノードクラスター、3 ノードクラスター、および標準クラスターに OpenShift Container Platform をデプロイすることをサポートします。

注記

すべてのマネージドクラスターには独自の namespace があり、**ManagedCluster** と **ClusterImageSet** を除くすべてのインストール CR はその namespace の下にあります。**ManagedCluster** と **ClusterImageSet** は、ネームスペーススコープではなく、クラスタースコープです。namespace および CR 名はクラスター名に一致します。

次の表に、設定した **SiteConfig** CR を使用してクラスターをインストールするときに RHACM アシストサービスによって自動的に適用されるインストール CR を示します。

表5.1 RHACM によって生成されたクラスターインストール CR

CR	説明	使用法
BareMetalHost	ターゲットのベアメタルホストの Baseboard Management Controller (BMC) の接続情報が含まれています。	Redfish プロトコルを使用して、BMC へのアクセスを提供し、ターゲットサーバーで検出イメージをロードおよび開始します。
InfraEnv	ターゲットのベアメタルホストに OpenShift Container Platform をインストールするための情報が含まれています。	**ClusterDeployment** で使用され、マネージドクラスターの Discovery ISO を生成します。
AgentClusterInstall	ネットワークやコントロールプレーンノードの数など、マネージドクラスター設定の詳細を指定します。インストールが完了すると、クラスター **kubeconfig** と認証情報が表示されます。	マネージドクラスターの設定情報を指定し、クラスターのインストール時にステータスを指定します。
ClusterDeployment	使用する **AgentClusterInstall** CR を参照します。	マネージドクラスターの Discovery ISO を生成するために **InfraEnv** で使用されます。

CR	説明	使用法
NMStateConfig	**MAC** アドレスから **IP** へのマッピング、DNS サーバー、デフォルトルート、およびその他のネットワーク設定などのネットワーク設定情報を提供します。	マネージドクラスターの Kube API サーバーの静的 IP アドレスを設定します。
Agent	ターゲットのベアメタルホストに関するハードウェア情報が含まれています。	ターゲットマシンの検出イメージの起動時にハブ上に自動的に作成されます。
ManagedCluster	クラスターがハブで管理されている場合は、インポートして知られている必要があります。この Kubernetes オブジェクトはそのインターフェイスを提供します。	ハブは、このリソースを使用してマネージドクラスターのステータスを管理し、表示します。
KlusterletAddonConfig	**ManagedCluster** リソースにデプロイされるハブによって提供されるサービスのリストが含まれます。	**ManagedCluster** リソースにデプロイするアドオンサービスをハブに指示します。
Namespace	ハブ上にある **ManagedCluster** リソースの論理領域。サイトごとに一意です。	リソースを **ManagedCluster** に伝搬します。
Secret	**BMC Secret** と **Image Pull Secret** の 2 つの CR が作成されます。	• **BMC Secret** は、ユーザー名とパスワードを使用して、ターゲットのベアメタルホストに対して認証を行います。 • **Image Pull Secre** には、ターゲットベアメタルホストにインストールされている OpenShift Container Platform イメージの認証情報が含まれます。
ClusterImageSet	リポジトリーおよびイメージ名などの OpenShift Container Platform イメージ情報が含まれます。	OpenShift Container Platform イメージを提供するためにリソースに渡されます。

第6章 VDU アプリケーションのワークロードに推奨される単一ノードの OPENSHIFT クラスター設定

以下の参照情報を使用して、仮想分散ユニット (vDU) アプリケーションをクラスターにデプロイするために必要な単一ノードの OpenShift 設定を理解してください。設定には、高性能ワークロードのためのクラスターの最適化、ワークロードの分割の有効化、およびインストール後に必要な再起動の回数の最小化が含まれます。

関連情報

- 単一クラスターを手動でデプロイするには、GitOps ZTP を使用した単一ノード OpenShift クラスターの手動インストール を参照してください。
- GitOps Zero Touch Provisioning (ZTP) を使用してクラスターのフリートをデプロイするには、GitOps ZTP を使用した遠端サイトのデプロイ を参照してください。

6.1. OPENSHIFT CONTAINER PLATFORM で低レイテンシーのアプリケーションを実行する

OpenShift Container Platform は、いくつかのテクノロジーと特殊なハードウェアデバイスを使用して、市販の (COTS) ハードウェアで実行するアプリケーションの低レイテンシー処理を可能にします。

RHCOS のリアルタイムカーネル

ワークロードが高レベルのプロセス決定で処理されるようにします。

CPU の分離

CPU スケジューリングの遅延を回避し、CPU 容量が一貫して利用可能な状態にします。

NUMA 対応のトポロジー管理

メモリーと Huge Page を CPU および PCI デバイスに合わせて、保証されたコンテナーメモリーと Huge Page を不均一メモリーアクセス (NUMA) ノードに固定します。すべての Quality of Service (QoS) クラスの Pod リソースは、同じ NUMA ノードに留まります。これにより、レイテンシーが短縮され、ノードのパフォーマンスが向上します。

Huge Page のメモリー管理

Huge Page サイズを使用すると、ページテーブルへのアクセスに必要なシステムリソースの量を減らすことで、システムパフォーマンスが向上します。

PTP を使用した精度同期

サブマイクロ秒の正確性を持つネットワーク内のノード間の同期を可能にします。

6.2. VDU アプリケーションワークロードに推奨されるクラスターホスト要件

vDU アプリケーションワークロードを実行するには、OpenShift Container Platform サービスおよび実稼働ワークロードを実行するのに十分なリソースを備えたベアメタルホストが必要です。

表6.1 最小リソース要件

プロファイル	vCPU	メモリー	ストレージ
最低限	4 ~ 8 個の vCPU コア	32GB のメモリー	120GB

注記

1 vCPU は、同時マルチスレッド (SMT) またはハイパースレッディングが有効にされていない場合に 1 つの物理コアと同等です。有効にした場合には、次の式を使用して対応する比率を計算します。

- (コアあたりのスレッド数×コア)×ソケット= vCPU

重要

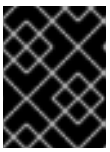
仮想メディアを使用して起動する場合は、サーバーには Baseboard Management Controller (BMC) が必要です。

6.3. 低遅延と高パフォーマンスのためのホストファームウェアの設定

ベアメタルホストでは、ホストをプロビジョニングする前にファームウェアを設定する必要があります。ファームウェアの設定は、特定のハードウェアおよびインストールの特定の要件によって異なります。

手順

1. **UEFI/BIOS Boot Mode** を **UEFI** に設定します。
2. ホスト起動シーケンスの順序で、**ハードドライブ** を設定します。
3. ハードウェアに特定のファームウェア設定を適用します。以下の表は、Intel FlexRAN 4G および 5G baseband PHY 参照設計をベースとした、Intel Xeon Skylake サーバーおよびそれ以降のハードウェア生成の典型的なファームウェア設定を説明しています。



重要

ファームウェア設定は、実際のハードウェアおよびネットワークの要件によって異なります。以下の設定例は、説明のみを目的としています。

表6.2 ファームウェア設定の例

ファームウェア設定	設定
CPU パワーとパフォーマンスポリシー	パフォーマンス
Uncore Frequency Scaling	Disabled
パフォーマンスの制限	Disabled
Intel SpeedStep ® Tech の強化	有効
Intel Configurable TDP	有効
設定可能な TDP レベル	レベル 2
Intel® Turbo Boost Technology	有効

ファームウェア設定	設定
energy Efficient Turbo	Disabled
Hardware P-States	Disabled
Package C-State	C0/C1 の状態
C1E	Disabled
Processor C6	Disabled

注記

ホストのファームウェアでグローバル SR-IOV および VT-d 設定を有効にします。これらの設定は、ベアメタル環境に関連します。

6.4. マネージドクラスターネットワークの接続の前提条件

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してマネージドクラスターをインストールおよびプロビジョニングするには、マネージドクラスターホストが次のネットワーク前提条件を満たしている必要があります。

- ハブクラスター内の GitOps ZTP コンテナーとターゲットベアメタルホストの Baseboard Management Controller (BMC) の間に双方向接続が必要です。
- マネージドクラスターは、ハブホスト名と ***.apps** ホスト名の API ホスト名を解決して到達できる必要があります。ハブの API ホスト名と ***.apps** ホスト名の例を次に示します。
  - **api.hub-cluster.internal.domain.com**
  - **console-openshift-console.apps.hub-cluster.internal.domain.com**
- ハブクラスターは、マネージドクラスターの API および ***.apps** ホスト名を解決して到達できる必要があります。マネージドクラスターの API ホスト名と ***.apps** ホスト名の例を次に示します。
  - **api.sno-managed-cluster-1.internal.domain.com**
  - **console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com**

6.5. GITOPS ZTP を使用した単一ノードの OPENSHIFT でのワークロードの分割

ワークロードのパーティショニングは、OpenShift Container Platform サービス、クラスター管理ワークロード、およびインフラストラクチャー Pod を、予約された数のホスト CPU で実行するように設定します。

GitOps Zero Touch Provisioning (ZTP) を使用してワークロードパーティショニングを設定するには、クラスターのインストールに使用する **SiteConfig** カスタムリソース (CR) の **cpuPartitioningMode** フィールドを設定し、ホスト上で **isolated** と **reserved** CPU を設定する **PerformanceProfile** CR を適

用します。

SiteConfig CR を設定すると、クラスターのインストール時にワークロードパーティショニングが有効になり、**PerformanceProfile** CR を適用すると、reserved および isolated セットへの割り当てが設定されます。これらの手順は両方とも、クラスターのプロビジョニング中に別々のタイミングで実行されます。

注記

SiteConfig CR の **cpuPartitioningMode** フィールドを使用したワークロードパーティショニングの設定は、OpenShift Container Platform 4.13 のテクノロジープレビュー機能です。

もしくは、**SiteConfig** カスタムリソース (CR) の **cpuset** フィールドとグループ **PolicyGenTemplate** CR の **reserved** フィールドを使用してクラスター管理 CPU リソースを指定できます。GitOps ZTP パイプラインは、これらの値を使用して、単一ノードの OpenShift クラスターを設定するワークロードパーティショニング **MachineConfig** CR (**cpuset**) および **PerformanceProfile** CR (**reserved**) の必須フィールドにデータを入力します。このメソッドは、OpenShift Container Platform 4.14 で一般公開された機能です。

ワークロードパーティショニング設定は、OpenShift Container Platform インフラストラクチャー Pod を **reserved** CPU セットに固定します。systemd、CRI-O、kubelet などのプラットフォームサービスは、**reserved** CPU セット上で実行されます。**isolated** CPU セットは、コンテナーワークロードに排他的に割り当てられます。CPU を分離すると、同じノード上で実行されている他のアプリケーションと競合することなく、ワークロードが指定された CPU に確実にアクセスできるようになります。分離されていないすべての CPU を予約する必要があります。

重要

reserved CPU セットと **isolated** CPU セットが重複しないようにしてください。

関連情報

- 推奨される単一ノードの OpenShift ワークロードパーティショニング設定については、ワークロードパーティショニング を参照してください。

6.6. 推奨されるクラスターインストールマニフェスト

ZTP パイプラインは、クラスターのインストール中に次のカスタムリソース (CR) を適用します。これらの設定 CR により、クラスターが vDU アプリケーションの実行に必要な機能とパフォーマンスの要件を満たしていることが保証されます。

注記

クラスターデプロイメントに GitOps ZTP プラグインと **SiteConfig** CR を使用する場合は、デフォルトで次の **MachineConfig** CR が含まれます。

デフォルトで含まれる CR を変更するには、**SiteConfig** の **extraManifests** フィルターを使用します。詳細は、SiteConfig CR を使用した高度なマネージドクラスター設定 を参照してください。

6.6.1. ワークロードの分割

DU ワークロードを実行する単一ノードの OpenShift クラスターには、ワークロードの分割が必要です。これにより、プラットフォームサービスの実行が許可されるコアが制限され、アプリケーションペイロードの CPU コアが最大化されます。

注記

ワークロードの分割は、クラスターのインストール中にのみ有効にできます。インストール後にワークロードパーティショニングを無効にすることはできません。ただし、**PerformanceProfile** CR を通じて、isolated セットと reserved セットに割り当てられた CPU のセットを変更できます。CPU 設定を変更すると、ノードが再起動します。

OPENSHIFT CONTAINER PLATFORM 4.12 から 4.13 以降への移行

ワークロードパーティショニングを有効にするために **cpuPartitioningMode** の使用に移行する場合は、クラスターのプロビジョニングに使用する **/extra-manifest** フォルダーからワークロードパーティショニングの **MachineConfig** CR を削除します。

ワークロードパーティショニング用に推奨される SiteConfig CR 設定

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "<site_name>"
  namespace: "<site_name>"
spec:
  baseDomain: "example.com"
  cpuPartitioningMode: AllNodes (1)
```

(1) クラスター内におけるすべてのノードのワークロードパーティショニングを設定するには、**cpuPartitioningMode** フィールドを **AllNodes** に設定します。

検証

アプリケーションとクラスターシステムの CPU ピニングが正しいことを確認します。以下のコマンドを実行します。

1. マネージドクラスターへのリモートシェルプロンプトを開きます。

```
$ oc debug node/example-sno-1
```

2. OpenShift インフラストラクチャーアプリケーションの CPU ピニングが正しいことを確認します。

```
sh-4.4# pgrep ovn | while read i; do taskset -cp $i; done
```

出力例

```
pid 8481's current affinity list: 0-1,52-53
pid 8726's current affinity list: 0-1,52-53
pid 9088's current affinity list: 0-1,52-53
pid 9945's current affinity list: 0-1,52-53
```

```
pid 10387's current affinity list: 0-1,52-53
pid 12123's current affinity list: 0-1,52-53
pid 13313's current affinity list: 0-1,52-53
```

3. システムアプリケーションの CPU ピニングが正しいことを確認します。

```
sh-4.4# pgrep systemd | while read i; do taskset -cp $i; done
```

出力例

```
pid 1's current affinity list: 0-1,52-53
pid 938's current affinity list: 0-1,52-53
pid 962's current affinity list: 0-1,52-53
pid 1197's current affinity list: 0-1,52-53
```

6.6.2. プラットフォーム管理フットプリントの削減

プラットフォームの全体的な管理フットプリントを削減するには、ホストオペレーティングシステムとは別の新しい namespace にすべての Kubernetes 固有のマウントポイントを配置する **MachineConfig** カスタムリソース (CR) が必要です。次の base64 でエンコードされた **MachineConfig** CR の例は、この設定を示しています。

推奨されるコンテナーマウント namespace 設定 (01-container-mount-ns-and-kubelet-conf-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKCmRlYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkgewogIGVj
aG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDApIFVOSVQgW2VudmZpbGUgW3Zhcm5hbWVdXQogIGV
jaG8KICBlY2hvIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3RhcnQgc3Rhbn
phIGZyb20gdGhlIGdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuIGl0IHRvIHN0ZG91dAogIGVj
aG8KICBlY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZW
FkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGV
mYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUIGlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxC
n0KCIVOSVQ9JDEKRU5WRklMRT0kMgpWQVJOQU1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUI
D09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCIgXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cm
FjdGluZyBFeGVjU3RhcnQgZnJvbSAkVU5JVCIKRklMRT0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCB
oZWFkIC1uIDEpCkZJTEU9JHtGSUxFI1wjIH0KaWYgW1sgISAtZiAkRklMRSBdXTsgdGhlbgogIGRlYnV
nICJGYWlsZWQgdG8gZmluZCByb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSIKICBle
Gl0CmZpCmRlYnVnICJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0k
KHNlZCAtbiAtZSAnL15FeGVjU3RhcnQ9LipcXCQvIHsgcy9eRXhlY1N0YXJ0PS8vOyBwIH0nIC1lICcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHAgfScgJEZJTEUp
```

```
CgppZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUlR9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogIGVjaG8gJEVYRUNTVEFSVApmaQo=
          mode: 493
          path: /usr/local/bin/extractExecStart
        - contents:
            source: data:text/plain;charset=utf-8;base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbWVzcGFjZS9tbnQgIiRAIgo=
          mode: 493
          path: /usr/local/bin/nsenterCmns
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Manages a mount namespace that both kubelet and crio can use to share their container-specific mounts

            [Service]
            Type=oneshot
            RemainAfterExit=yes
            RuntimeDirectory=container-mount-namespace
            Environment=RUNTIME_DIRECTORY=%t/container-mount-namespace
            Environment=BIND_POINT=%t/container-mount-namespace/mnt
            ExecStartPre=bash -c "findmnt ${RUNTIME_DIRECTORY} || mount --make-unbindable --bind ${RUNTIME_DIRECTORY} ${RUNTIME_DIRECTORY}"
            ExecStartPre=touch ${BIND_POINT}
            ExecStart=unshare --mount=${BIND_POINT} --propagation slave mount --make-rshared /
            ExecStop=umount -R ${RUNTIME_DIRECTORY}
          name: container-mount-namespace.service
        - dropins:
            - contents: |
                [Unit]
                Wants=container-mount-namespace.service
                After=container-mount-namespace.service

                [Service]
                ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
                EnvironmentFile=-/%t/%N-execstart.env
                ExecStart=
                ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
                    ${ORIG_EXECSTART}"
              name: 90-container-mount-namespace.conf
          name: crio.service
        - dropins:
            - contents: |
                [Unit]
                Wants=container-mount-namespace.service
                After=container-mount-namespace.service

                [Service]
                ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
                EnvironmentFile=-/%t/%N-execstart.env
                ExecStart=
```

```
        ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
          ${ORIG_EXECSTART} --housekeeping-interval=30s"
      name: 90-container-mount-namespace.conf
    - contents: |
        [Service]
        Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
        Environment="OPENSHIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
      name: 30-kubelet-interval-tuning.conf
  name: kubelet.service
```

6.6.3. SCTP

Stream Control Transmission Protocol (SCTP) は、RAN アプリケーションで使用される主要なプロトコルです。この **MachineConfig** オブジェクトは、SCTP カーネルモジュールをノードに追加して、このプロトコルを有効にします。

推奨されるコントロールプレーンノードの SCTP 設定 (03-sctp-machine-config-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module-master
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf
```

推奨されるワーカーノードの SCTP 設定 (03-sctp-machine-config-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: load-sctp-module-worker
spec:
  config:
    ignition:
      version: 2.2.0
```

```
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf
```

6.6.4. rcu_normal の設定

次の **MachineConfig** CR は、システムの起動完了後に **rcu_normal** を 1 に設定するようにシステムを設定します。これにより、vDU アプリケーションのカーネル遅延が改善されます。

ノードの起動完了後に rcu_expedited を無効にするために推奨される設定 (08-set-rcu-normal-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 08-set-rcu-normal-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKIwojIERpc2FibGUgcmN1X2V4cGVkaXRlZCBhZnRlciBub2RlIGhhcyBmaW
5pc2hlZCBib290aW5nCiMKIyBUaGUgZGVmYXVsdHMgYmVsb3cgY2FuIGJlIG92ZXJyaWRkZW4gdml
hIGVudmlyb25tZW50IHZhcmlhYmxlcwojCgojIERlZmF1bHQgd2FpdCB0aW1lIGlzIDYwMHMgPSAxMG
06Ck1BWElNVU1fV0FJVF9USU1FPSR7TUFYSU1VTV9XQUlUX1RJTUU6LTYwMH0KCiMgRGVmY
XVsdCBzdGVhZHktc3RhdGUgdGhyZXNob2xkID0gMiUKIyBBbGxvd2VkIHZhbHVlczoKIyAgNCAgLSBh
YnNvbHV0ZSBwb2QgY291bnQgKCsvLSkKIyAgNCUgLSBwZXJjZW50IGNoYW5nZSAoKy8tKSBhY2N1
MSAtIGRpc2FibGUgdGhlIHN0ZWFkeS1zdGF0ZSBjaGVjawpTVEVBRFlfU1RBVEVfVEhSRVNIT0xEP
SR7U1RFQURZX1NUQVRFX1RIUkVTSE9MRDotMiV9CgojIERlZmF1bHQgc3RlYWR5LXN0YXRlIHd
pbmRvdyA9IDYwcwojIElmIHRoZSBydW5uaW5nIHBvZCBjb3VudCBzdGF5cyB3aXRoaW4gdGhlIGdpd
mVuIHRocmVzaG9sZCBmb3IgdGhpcyB0aW1lCiMgcGVyaW9kLCByZXR1cm4gQ1BVIHV0aWxpemF0
aW9uIHRvIG5vcm1hbCBiZWZvcmUgdGhlIG1heGltdW0gd2FpdCB0aW1lIHRvcHVsc3RlYWR5LXN0YX
RFQURZX1NUQVRFX1dJTkRPVz0ke1NURUFEWV9TVEFURV9XSU5ET1c6LTYwfQoKIyBEZWZhd
Wx0IHN0ZWFkeS1zdGF0ZSBhbGxvd3MgYW55IHBvZCBjb3VudCB0byBiZSAic3RlYWR5IHN0YXRlIgo
jIEluY3JlYXNpbmcgdGhpcyB3aWxsIHNraXAgYW55IHN0ZWFkeS1zdGF0ZSBjaGVja3MgdW50aWwgd
GhlIGNvdW50IHJpc2VzIGFib3ZlCiMgdGhpcyBudW1iZXIgdG8gYXZaWQgZmFsc2UgcG9zaXRpdmV
zIGlmIHRoZXJlIGFyZSBzb21lIHBlcmlvZHMgd2hlcmUgdGhlCiMgY291bnQgZG9lc24ndCBpbmNyZWF
zZSBidXQgd2Uga25vdyB3ZSBjYW4ndCBiZSBhdCBzdGVhZHktc3RhdGUgeWV0LgpTVEVBRFlfU1RB
VEVfTUlOSU1VTT0ke1NURUFEWV9TVEFURV9NSU5JTVVNOi0wfQoKIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIwoKd2l0aGluKCkgewogIGxvY2FsIGxhc3Q9JDEgY3Vycm
```

```
udD0kMiB0aHJlc2hvbGQ9JDMKICBsb2NhbCBkZWx0YT0wIHBjaGFuZ2UKICBkZWx0YT0kKCggY3Vy
cmVudCAtIGxhc3QgKSkKICBpZiBbWyAkY3VycmVudCAtZXEgJGxhc3QgXV07IHRoZW4KICAgIHBjaG
FuZ2U9MAogIGVsaWYgW1sgJGxhc3QgLWVxIDAgXV07IHRoZW4KICAgIHBjaGFuZ2U9MTAwMDA
wMAogIGVsc2UKICAgIHBjaGFuZ2U9JCgoICggIiRkZWx0YSIgKiAxMDApIC8gbGFzdCApKQogIGZpCi
AgZWNobyAtbiAibGFzdDokbGFzdCBjdXJyZW50OiRjdXJyZW50IGRlbHRhOiRkZWx0YSBwY2hhbmdl
OiR7cGNoYW5nZX0lOiAiCiAgbG9jYWwgcYWJzb2x1dGUgbGltaXQKICBjYXNlICRjaGFuZ2UgaW4
KICAgIColKQogICAgICBhYnNvbHV0ZT0ke3BjaGFuZ2UjIy19CiAgICAgIDs7CiAgICBlc2FjCiAg
ICAgIGFic29sdXRlIHC1sZSAkbGltaXQgXV07IHRoZW4KICAgIGVjaG8gIndpdGhpbiAoKyB8tKSR
0aHJlc2hvbGQiCiAgICByZXR1cm4gMAogIGVsc2UKICAgIGVjaG8gIm91dHNpZGUgKCsvLSkkdGhyZ
XNob2xkIgogICAgcmV0dXJuIDEKICBmaQp9CgpzdGVhZHlzdGF0ZSgpIHsKICBsb2NhbCBsYXN0PS
QxIGN1cnJlbnQ9JDIKICBpZiBbWyAkbHQgJFNURUFEWV9TVEFURV9NSU5JTVVNIF1dO
yB0aGVuCiAgICBlY2hvICJsYXN0OiRsYXN0IGN1cnJlbnQ6JGN1cnJlbnQgV2FpdGluZyB0byByZWFja
CAkU1RFQURZX1NUQVRFX01JTklNVU0gYmVmb3JlIGNoZWNraW5nIGZvciBzdGVhZHktc3RhdGUi
CiAgICByZXR1cm4gMQogIGZpCiAgd2l0aGluICIkbGFzdCIgIiRjdXJyZW50IiAiJFNURUFEWV9TVEFU
RV9USFJFU0hPTEQiCn0KCndhaXRGb3JSZWFkeSgpIHsKICBsb2dnZXIgIlJlY292ZXJ5OiBXYWl0aW
5nICR7TUFYSU1VTV9XQUlUX1RJTUV9cyBmb3IgdGhlIGluXRpYWxpemF0aW9uIHRvIGNvbXBsZX
RlIgogIGxvY2FsIHQ9MCBzPTEwCiAgbG9jYWwgbGFzdENjb3VudD0wIGNjb3VudD0wIHN0ZWFkeVN
0YXRlVGltZT0wCiAgd2hpbGUgW1sgJHQgLWx0ICRNQVhJTVVNX1dBSVRfVElNRSBdXTsgZG8KIC
AgIHNsZWVwICRzCiAgICAoKHQgKz0gcykpCiAgICBjdXJcdCmljdGwgcHMgMj4vZGV2L251bGwgfCB3YyAtbCkKICAgIC
W50CiAgICBjY291bnQ9JChjcmljdGwgcHMgMj4vZGV2L251bGwgfCB3YyAtbCkKICAgIGlmIFtbICRjY2
91bnQgLWd0IDAgXV0gJiYgc3RlYWR5c3RhdGUgIiRsYXN0Q2NvdW50IiAiJGNjb3VudCI7IHRoZW4KI
CAgICAgKChzdGVhZHlTdGF0ZVRpbWUgKz0gcykpCiAgICAgIGVjaG8gIlN0ZWFkeS1zdGF0ZSBmb3
IgJHtzdGVhZHlTdGF0ZVRpbWV9cy8ke1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgaWYgW1
1sgJHN0ZWFkeVN0YXRlVGltZSAtZ2UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICA
gICAgICBsb2dnZXIgIlJlY292ZXJ5OiBTdGVhZHktc3RhdGUgKCsvLSAkU1RFQURZX1NUQVRFX1RI
UkVTSE9MRCkgZm9yICR7U1RFQURZX1NUQVRFX1dJTkRPV31zOiBEb25lIgogICAgICAgIHN0ZWFkeV
biAwCiAgICAgICAgZXhpdCAwCiAgICAgIGZpCiAgICBlbHNlCiAgICAgIGlmIFtbICRzdGVhZHlTdGF0ZVRpbWUgLWd0IDAgXV07
IHRoZW4KICAgICAgICBlY2hvICJSZXNldHRpbmcgc3RlYWR5LXN0YXRlIHRpbWVyIgogICAgICAgIHN
0ZWFkeVN0YXRlVGltZT0wCiAgICAgICAgZmkKICAgICAgcHMgMj4vZGV2L251bGwgfCB3YyAtbCkKICAgZ
G9uZQogIGxvZ2dlciAiUmVjb3Zlcnk6IFJlY292ZXJ5IENvbXBsZXRlIFRpbWVvdXQiCn0KCnNldFJjdU5vc
m1hbCgpIHsKICBlY2hvICJTZXR0aW5nIHJdV9ub3JtYWwgdG8gMSIKICBlY2hvIDEgPiAvc3lzL2tlcm5lb
C9yY3Vfbm9ybWFsCn0KCm1haW4oKSB7CiAgd2FpdEZvclJlYWR5CiAgZWNobyAiV2FpdGluZyBmb3I
gc3RlYWR5IHN0YXRlIHRvb2s6ICQoYXdrICd7cHJpbnQgaW50KCQxLzM2MDApImgiLCBpbnQoKCQ
xJTM2MDApLzYwKSJtIiwgaW50KCQxJTYwKSJzIn0nIC9wcm9jL3VwdGltZSkiCiAgc2V0UmN1Tm9yb
WFsCn0KCmlmIFtbICIke0BBU0hfU09VUkNFWzBdfSIgPSAiJHswfSIgXV07IHRoZW4KICBtYWluICIke0
B9IgogIGV4aXQgJD8KZmkK
        mode: 493
        path: /usr/local/bin/set-rcu-normal.sh
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Disable rcu_expedited after node has finished booting by setting rcu_normal to 1

            [Service]
            Type=simple
            ExecStart=/usr/local/bin/set-rcu-normal.sh

            # Maximum wait time is 600s = 10m:
            Environment=MAXIMUM_WAIT_TIME=600

            # Steady-state threshold = 2%
            # Allowed values:
            #  4  - absolute pod count (+/-)
```

```
          #  4% - percent change (+/-)
          #  -1 - disable the steady-state check
          # Note: '%' must be escaped as '%%' in systemd unit files
          Environment=STEADY_STATE_THRESHOLD=2%%

          # Steady-state window = 120s
          # If the running pod count stays within the given threshold for this time
          # period, return CPU utilization to normal before the maximum wait time has
          # expires
          Environment=STEADY_STATE_WINDOW=120

          # Steady-state minimum = 40
          # Increasing this will skip any steady-state checks until the count rises above
          # this number to avoid false positives if there are some periods where the
          # count doesn't increase but we know we can't be at steady-state yet.
          Environment=STEADY_STATE_MINIMUM=40

          [Install]
          WantedBy=multi-user.target
        enabled: true
        name: set-rcu-normal.service
```

6.6.5. kdump による自動カーネルクラッシュダンプ

kdump は、カーネルがクラッシュしたときにカーネルクラッシュダンプを作成する Linux カーネル機能です。**kdump** は、次の **MachineConfig** CR で有効になっています。

コントロールプレーンの kdump ログから Ice ドライバーを削除するために推奨される MachineConfig CR (05-kdump-config-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 05-kdump-config-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
    storage:
```

```
    files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2UgbW9k
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWluIHNlcnZlcnMuCi
MgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9li91c3Iv
YmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcgcHVycG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iCiJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZiBma WxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mICR
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBma WxlIGFscmVhZHkgdXBkYXRlZAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKIyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcgbGlrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucl9jcHVzPTEgLi4uIGhlc3RfZGlzYWJsZSIKIyBVc2Ugc2VkIHRvIG1hdG
NoIGV2ZXJ5dGhpbmcgYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7U0VEfSAtaSAncy9eKEtEVU1QX0NPTU1BTkRMSU5FX0FQUEVORD0iLiovXDE
gJHtSRU1PVkVfSUNFX1NUUn0vJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
        mode: 448
        path: /usr/local/bin/kdump-remove-ice-module.sh
```

コントロールプレーンノード用に推奨される kdump 設定 (06-kdump-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 06-kdump-enable-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

ワーカーノードの kdump ログから ice ドライバーを削除するために推奨される MachineConfig CR (05-kdump-config-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-kdump-config-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
```

```
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKIyBUaGlzIHNjcmlwdCByZW1vdmVzIHRoZSBpY2UgbW9k
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrZHVtcCBmYWlsdXJlcyBvbiBjZXJ0YWluIHNlcnZlcnMuCi
MgVGhpcyBpcyBhIHRlbXBvcmFyeSB3b3JrYXJvdW5kIGZvciBSSEVMUExBTi0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVkIHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9li91c3Iv
YmluL3NlZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcgcHVycG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iClJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZiBmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mIC
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxlIGFscmVhZHkgdXBkYXRlZAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKIyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcgbGlrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucl9jcHVzPTEgLi4uIGhlc3RfZGlzYWJsZSIKIyBVc2Ugc2VkIHRvIG1hdG
NoIGV2ZXJ5dGhpbmcgYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlIFJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7S0RVTVBfQ09ORn0gLWUgIi9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPS8iJdKi8m
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
          mode: 448
          path: /usr/local/bin/kdump-remove-ice-module.sh
```

kdump ワーカーノード用に推奨される設定 (06-kdump-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 06-kdump-enable-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
  kernelArguments:
    - crashkernel=512M
```

6.6.6. CRI-O キャッシュの自動ワイプを無効にする

制御されていないホストのシャットダウンまたはクラスターの再起動の後、CRI-O は CRI-O キャッシュ全体を自動的に削除します。そのため、ノードの再起動時にはすべてのイメージがレジストリーか

らプルされます。これにより、許容できないほど復元に時間がかかったり、復元が失敗したりする可能性があります。GitOps ZTP を使用してインストールするシングルノード OpenShift クラスターでこの問題が発生しないようにするには、クラスターをインストールする際に CRI-O 削除キャッシュ機能を無効にします。

コントロールプレーンノードで CRI-O キャッシュワイプを無効にするために推奨される MachineConfig CR (99-crio-disable-wipe-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-crio-disable-wipe-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAiIgo=
          mode: 420
          path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

ワーカーノードで CRI-O キャッシュワイプを無効にするために推奨される MachineConfig CR (99-crio-disable-wipe-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-crio-disable-wipe-worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAiIgo=
          mode: 420
          path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

6.6.7. crun をデフォルトのコンテナーランタイムに設定

次の **ContainerRuntimeConfig** カスタムリソース (CR) は、コントロールプレーンおよびワーカーノードのデフォルト OCI コンテナーランタイムとして crun を設定します。crun コンテナーランタイムは高速かつ軽量で、メモリーフットプリントも小さくなります。

重要

パフォーマンスを最適化するには、シングルノード OpenShift、3 ノード OpenShift、および標準クラスターのコントロールプレーンとワーカーノードで crun を有効にします。CR 適用時にクラスターが再起動するのを回避するには、GitOps ZTP の追加の Day 0 インストール時マニフェストとして変更を適用します。

コントロールプレーンノード用に推奨される ContainerRuntimeConfig (enable-crun-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: enable-crun-master
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/master: ""
  containerRuntimeConfig:
    defaultRuntime: crun
```

ワーカーノード用に推奨される ContainerRuntimeConfig (enable-crun-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: enable-crun-worker
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  containerRuntimeConfig:
    defaultRuntime: crun
```

6.7. 推奨されるインストール後のクラスター設定

クラスターのインストールが完了すると、ZTP パイプラインは、DU ワークロードを実行するために必要な次のカスタムリソース (CR) を適用します。

注記

GitOps ZTP v4.10 以前では、**MachineConfig** CR を使用して UEFI セキュアブートを設定します。これは、GitOps ZTP v4.11 以降では不要になりました。v4.11 では、クラスターのインストールに使用する **SiteConfig** CR の **spec.clusters.nodes.bootMode** フィールドを更新することで、単一ノード OpenShift クラスターの UEFI セキュアブートを設定します。詳細は、SiteConfig および GitOps ZTP を使用したマネージドクラスターのデプロイ を参照してください。

6.7.1. Operator

DU ワークロードを実行するシングルノード OpenShift クラスターには、次の Operator をインストールする必要があります。

- Local Storage Operator
- Logging Operator
- PTP Operator
- SR-IOV Network Operator

カスタム **CatalogSource** CR を設定し、デフォルトの **OperatorHub** 設定を無効にし、インストールするクラスターからアクセスできる **ImageContentSourcePolicy** ミラーレジストリーを設定する必要もあります。

推奨される Storage Operator namespace と Operator グループ設定 (StorageNS.yaml、StorageOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-local-storage
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-local-storage
  namespace: openshift-local-storage
  annotations: {}
spec:
  targetNamespaces:
    - openshift-local-storage
```

推奨される Cluster Logging Operator namespace と Operator グループの設定 (ClusterLogNS.yaml、ClusterLogOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations: {}
spec:
  targetNamespaces:
    - openshift-logging
```

推奨される PTP Operator namespace と Operator グループ設定 (PtpSubscriptionNS.yaml、 PtpSubscriptionOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
  annotations: {}
spec:
  targetNamespaces:
    - openshift-ptp
```

推奨される SR-IOV Operator namespace と Operator グループ設定 (SriovSubscriptionNS.yaml、 SriovSubscriptionOperGroup.yaml)

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  targetNamespaces:
    - openshift-sriov-network-operator
```

推奨される CatalogSource 設定 (DefaultCatsrc.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: default-cat-source
  namespace: openshift-marketplace
  annotations:
    target.workload.openshift.io/management: '{"effect": "PreferredDuringScheduling"}'
spec:
```

```
  displayName: default-cat-source
  image: $imageUrl
  publisher: Red Hat
  sourceType: grpc
  updateStrategy:
    registryPoll:
      interval: 1h
status:
  connectionState:
    lastObservedState: READY
```

推奨される ImageContentSourcePolicy 設定 (DisconnectedICSP.yaml)

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: disconnected-internal-icsp
  annotations: {}
spec:
  repositoryDigestMirrors:
    - $mirrors
```

推奨される OperatorHub 設定 (OperatorHub.yaml)

```
apiVersion: config.openshift.io/v1
kind: OperatorHub
metadata:
  name: cluster
  annotations: {}
spec:
  disableAllDefaultSources: true
```

6.7.2. Operator のサブスクリプション

DU ワークロードを実行する単一ノードの OpenShift クラスターには、次の **Subscription** CR が必要です。サブスクリプションは、次の Operator をダウンロードする場所を提供します。

- Local Storage Operator
- Logging Operator
- PTP Operator
- SR-IOV Network Operator
- SRIOV-FEC Operator

Operator サブスクリプションごとに、Operator の取得先であるチャネルを指定します。推奨チャンネルは **stable** です。

Manual 更新または **Automatic** 更新を指定できます。**Automatic** モードでは、Operator は、レジストリーで利用可能になると、チャネル内の最新バージョンに自動的に更新します。**Manual** モードでは、新しい Operator バージョンは、明示的に承認された場合にのみインストールされます。

ヒント

サブスクリプションには **Manual** モードを使用します。これにより、スケジュールされたメンテナンス期間内に収まるように Operator の更新タイミングを制御できます。

推奨される Local Storage Operator サブスクリプション (StorageSubscription.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
  annotations: {}
spec:
  channel: "stable"
  name: local-storage-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推奨される SR-IOV Operator サブスクリプション (SriovSubscription.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  channel: "stable"
  name: sriov-network-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推奨される PTP Operator サブスクリプション (PtpSubscription.yaml)

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
  annotations: {}
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
```

```
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

推奨される Cluster Logging Operator サブスクリプション (ClusterLogSubscription.yaml)

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations: {}
spec:
  channel: "stable"
  name: cluster-logging
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

6.7.3. クラスターのロギングとログ転送

DU ワークロードを実行する単一ノードの OpenShift クラスターでは、デバッグのためにロギングとログ転送が必要です。次の **ClusterLogging** および **ClusterLogForwarder** カスタムリソース (CR) が必要です。

推奨されるクラスターロギング設定 (ClusterLogging.yaml)

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
  annotations: {}
spec:
  managementState: "Managed"
  collection:
    logs:
      type: "vector"
```

推奨されるログ転送設定 (ClusterLogForwarder.yaml)

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
  annotations: {}
spec:
  outputs: $outputs
  pipelines: $pipelines
```

spec.outputs.url フィールドを、ログの転送先となる Kafka サーバーの URL に設定します。

6.7.4. パフォーマンスプロファイル

DU ワークロードを実行する単一ノードの OpenShift クラスターでは、リアルタイムのホスト機能とサービスを使用するために Node Tuning Operator パフォーマンスプロファイルが必要です。

注記

OpenShift Container Platform の以前のバージョンでは、パフォーマンスアドオン Operator を使用して自動チューニングを実装し、OpenShift アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

次の **PerformanceProfile** CR の例は、必要なシングルノード OpenShift クラスター設定を示しています。

推奨されるパフォーマンスプロファイル設定 (PerformanceProfile.yaml)

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
  # matches this name: include=openshift-node-performance-${PerformanceProfile.metadata.name}
  # Also in file 'validatorCRs/informDuValidator.yaml':
  # name: 50-performance-${PerformanceProfile.metadata.name}
  name: openshift-node-performance-profile
  annotations:
    ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "vfio_pci.enable_sriov=1"
    - "vfio_pci.disable_idle_d3=1"
    - "module_blacklist=irdma"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  # To use the standard (non-realtime) kernel, set enabled to false
  realTimeKernel:
    enabled: true
  workloadHints:
```

```
    # WorkloadHints defines the set of upper level flags for different type of workloads.
    # See https://github.com/openshift/cluster-node-tuning-operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
    # for detailed descriptions of each item.
    # The configuration below is set for a low latency, performance mode.
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false
```

表6.3 シングルノード OpenShift クラスターの PerformanceProfile CR オプション

PerformanceProfile CR フィールド	説明
metadata.name	**name** が、関連する GitOps ZTP カスタムリソース (CR) に設定されている次のフィールドと一致していることを確認してください。 • **TunedPerformancePatch.yaml** の **include=openshift-node-performance-${PerformanceProfile.metadata.name}** • **validatorCRs/informDuValidator.yaml** の **name: 50-performance-${PerformanceProfile.metadata.name}**
spec.additionalKernelArgs	**efi=runtime** は、クラスターホストの UEFI セキュアブートを設定します。
spec.cpu.isolated	分離された CPU を設定します。すべてのハイパースレッディングペアが一致していることを確認します。 **重要** 予約済みおよび分離された CPU プールは重複してはならず、いずれも使用可能なすべてのコア全体にわたる必要があります。考慮されていない CPU コアは、システムで未定義の動作を引き起こします。
spec.cpu.reserved	予約済みの CPU を設定します。ワークロードの分割が有効になっている場合、システムプロセス、カーネルスレッド、およびシステムコンテナースレッドは、これらの CPU に制限されます。分離されていないすべての CPU を予約する必要があります。

PerformanceProfile CR フィールド	説明
spec.hugepages.pages	• huge page の数 (**count**) を設定します。 • huge page のサイズ (**size**) を設定します。 • **node** を **hugepage** が割り当てられた NUMA ノード (**node**) に設定します。
spec.realTimeKernel	リアルタイムカーネルを使用するには、**enabled** を **true** に設定します。
spec.workloadHints	**workloadHints** を使用して、各種ワークロードの最上位フラグのセットを定義します。この例では、クラスターが低レイテンシーかつ高パフォーマンスになるように設定されています。

6.7.5. クラスター時間同期の設定

コントロールプレーンまたはワーカーノードに対して、1回限りのシステム時間同期ジョブを実行します。

コントロールプレーンノード用に推奨される1回限りの時間同期 (99-sync-time-once-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-sync-time-once-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Sync time once
            After=network-online.target
            Wants=network-online.target
            [Service]
            Type=oneshot
            TimeoutStartSec=300
            ExecCondition=/bin/bash -c 'systemctl is-enabled chronyd.service --quiet && exit 1 || exit 0'
            ExecStart=/usr/sbin/chronyd -n -f /etc/chrony.conf -q
            RemainAfterExit=yes
            [Install]
```

```
      WantedBy=multi-user.target
    enabled: true
    name: sync-time-once.service
```

ワーカーノード用に推奨される 1 回限りの時間同期 (99-sync-time-once-worker.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-sync-time-once-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Sync time once
            After=network-online.target
            [Service]
            Type=oneshot
            TimeoutStartSec=300
            ExecCondition=/bin/bash -c 'systemctl is-enabled chronyd.service --quiet && exit 1 || exit 0'
            ExecStart=/usr/sbin/chronyd -n -f /etc/chrony.conf -q
            RemainAfterExit=yes
            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: sync-time-once.service
```

6.7.6. PTP

単一ノードの OpenShift クラスターは、ネットワーク時間同期に Precision Time Protocol (PTP) を使用します。次の **PtpConfig** CR の例は、通常のクロック、境界クロック、およびグランドマスタークロックに必要な PTP 設定を示しています。適用する設定は、ノードのハードウェアとユースケースにより異なります。

推奨される PTP 通常クロック設定 (PtpConfigSlave.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: slave
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "slave"
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2 -s"
```

```
    phc2sysOpts: "-a -r -n 24"
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    ptp4lConf: |
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      slaveOnly 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 255
      clockAccuracy 0xFE
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
      dscp_event 0
      dscp_general 0
      dataset_comparison G.8275.x
      G.8275.defaultDS.localPriority 128
      #
      # Port Data Set
      #
      logAnnounceInterval -3
      logSyncInterval -4
      logMinDelayReqInterval -4
      logMinPdelayReqInterval -4
      announceReceiptTimeout 3
      syncReceiptTimeout 0
      delayAsymmetry 0
      fault_reset_interval -4
      neighborPropDelayThresh 20000000
      masterOnly 0
      G.8275.portDS.localPriority 128
      #
      # Run time options
      #
      assume_two_step 0
      logging_level 6
      path_trace_enabled 0
      follow_up_info 0
      hybrid_e2e 0
      inhibit_multicast_service 0
      net_sync_monitor 0
      tc_spanning_tree 0
      tx_timestamp_timeout 50
      unicast_listen 0
      unicast_master_table 0
      unicast_req_duration 3600
      use_syslog 1
      verbose 0
```

```
      summary_interval 0
      kernel_leap 1
      check_fup_sync 0
      clock_class_threshold 7
      #
      # Servo Options
      #
      pi_proportional_const 0.0
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
recommend:
  - profile: "slave"
    priority: 4
    match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"
```

推奨される境界クロック設定 (PtpConfigBoundary.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "boundary"
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        # The interface name is hardware-specific
        [$iface_slave]
        masterOnly 0
        [$iface_master_1]
        masterOnly 1
        [$iface_master_2]
        masterOnly 1
        [$iface_master_3]
        masterOnly 1
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
```

```
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
```

```
        delay_mechanism E2E
        time_stamping hardware
        tsproc_mode filter
        delay_filter moving_median
        delay_filter_length 10
        egressLatency 0
        ingressLatency 0
        boundary_clock_jbod 0
        #
        # Clock description
        #
        productDescription ;;
        revisionData ;;
        manufacturerIdentity 00:00:00
        userDescription ;
        timeSource 0xA0
  recommend:
    - profile: "boundary"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

推奨される PTP Westport Channel e810 グランドマスタークロック設定 (PtpConfigGmWpc.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4lOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffSet: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 100
          pins: $e810_pins
          #  "$iface_master":
          #    "U.FL2": "0 2"
          #    "U.FL1": "0 1"
          #    "SMA2": "0 2"
          #    "SMA1": "0 1"
          ublxCmds:
            - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
```

```
        - "-P"
        - "29.20"
        - "-z"
        - "CFG-HW-ANT_CFG_VOLTCTRL,1"
      reportOutput: false
    - args: #ubxtool -P 29.20 -e GPS
        - "-P"
        - "29.20"
        - "-e"
        - "GPS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d Galileo
        - "-P"
        - "29.20"
        - "-d"
        - "Galileo"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d GLONASS
        - "-P"
        - "29.20"
        - "-d"
        - "GLONASS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d BeiDou
        - "-P"
        - "29.20"
        - "-d"
        - "BeiDou"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
        - "-P"
        - "29.20"
        - "-t"
        - "-w"
        - "5"
        - "-v"
        - "1"
        - "-e"
        - "SURVEYIN,600,50000"
      reportOutput: true
    - args: #ubxtool -P 29.20 -p MON-HW
        - "-P"
        - "29.20"
        - "-p"
        - "MON-HW"
      reportOutput: true
  ts2phcOpts: " "
  ts2phcConf: |
    [nmea]
    ts2phc.master 1
```

```
    [global]
    use_syslog  0
    verbose 1
    logging_level 7
    ts2phc.pulsewidth 100000000
    #cat /dev/GNSS to find available serial port
    #example value of gnss_serialport is /dev/ttyGNSS_1700_0
    ts2phc.nmea_serialport $gnss_serialport
    leapfile  /usr/share/zoneinfo/leap-seconds.list
    [$iface_master]
    ts2phc.extts_polarity rising
    ts2phc.extts_correction 0
  ptp4lConf: |
    [$iface_master]
    masterOnly 1
    [$iface_master_1]
    masterOnly 1
    [$iface_master_2]
    masterOnly 1
    [$iface_master_3]
    masterOnly 1
    [global]
    #
    # Default Data Set
    #
    twoStepFlag 1
    priority1 128
    priority2 128
    domainNumber 24
    #utc_offset 37
    clockClass 6
    clockAccuracy 0x27
    offsetScaledLogVariance 0xFFFF
    free_running 0
    freq_est_interval 1
    dscp_event 0
    dscp_general 0
    dataset_comparison G.8275.x
    G.8275.defaultDS.localPriority 128
    #
    # Port Data Set
    #
    logAnnounceInterval -3
    logSyncInterval -4
    logMinDelayReqInterval -4
    logMinPdelayReqInterval 0
    announceReceiptTimeout 3
    syncReceiptTimeout 0
    delayAsymmetry 0
    fault_reset_interval -4
    neighborPropDelayThresh 20000000
    masterOnly 0
    G.8275.portDS.localPriority 128
    #
    # Run time options
    #
```

```
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit  200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
```

```
        #
        # Clock description
        #
        productDescription ;;
        revisionData ;;
        manufacturerIdentity 00:00:00
        userDescription ;
        timeSource 0x20
  recommend:
    - profile: "grandmaster"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

次のオプションの **PtpOperatorConfig** CR は、ノードの PTP イベントレポートを設定します。

推奨される PTP イベント設定 (PtpOperatorConfigForEvent.yaml)

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
  annotations: {}
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/$mcp: ""
  ptpEventConfig:
    enableEventPublisher: true
    transportHost: "http://ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
```

6.7.7. 拡張調整済みプロファイル

DU ワークロードを実行する単一ノードの OpenShift クラスターには、高性能ワークロードに必要な追加のパフォーマンスチューニング設定が必要です。次の **Tuned** CR の例では、**Tuned** プロファイルを拡張しています。

推奨される拡張 Tuned プロファイル設定 (TunedPerformancePatch.yaml)

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: performance-patch
      # Please note:
      # - The 'include' line must match the associated PerformanceProfile name, following below
pattern
      #   include=openshift-node-performance-${PerformanceProfile.metadata.name}
```

```
    # - When using the standard (non-realtime) kernel, remove the kernel.timer_migration override
from
    #   the [sysctl] section and remove the entire section if it is empty.
    data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
      [scheduler]
      group.ice-ptp=0:f:10:*:ice-ptp.*
      group.ice-gnss=0:f:10:*:ice-gnss.*
      group.ice-dplls=0:f:10:*:ice-dplls.*
      [service]
      service.stalld=start,enable
      service.chronyd=stop,disable
  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: "$mcp"
      priority: 19
      profile: performance-patch
```

表6.4 シングルノード OpenShift クラスター用の Tuned CR オプション

調整された CR フィールド	説明
spec.profile.data	• **spec.profile.data** で設定する **include** 行は、関連付けられた **PerformanceProfile** CR 名と一致する必要があります。たとえば **include=openshift-node-performance-${PerformanceProfile.metadata.name}** です。

6.7.8. SR-IOV

シングルルート I/O 仮想化 (SR-IOV) は、一般的にフロントホールネットワークとミッドホールネットワークを有効にするために使用されます。次の YAML の例では、単一ノードの OpenShift クラスターの SR-IOV を設定します。

注記

SriovNetwork CR の設定は、特定のネットワークとインフラストラクチャーの要件によって異なります。

推奨される SriovOperatorConfig CR 設定 (SriovOperatorConfig.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
```

```
  configDaemonNodeSelector:
    "node-role.kubernetes.io/$mcp": ""
  # Injector and OperatorWebhook pods can be disabled (set to "false") below
  # to reduce the number of management pods. It is recommended to start with the
  # webhook and injector pods enabled, and only disable them after verifying the
  # correctness of user manifests.
  #   If the injector is disabled, containers using sr-iov resources must explicitly assign
  #   them in the  "requests"/"limits" section of the container spec, for example:
  #    containers:
  #    - name: my-sriov-workload-container
  #      resources:
  #        limits:
  #          openshift.io/<resource_name>:  "1"
  #        requests:
  #          openshift.io/<resource_name>:  "1"
  enableInjector: false
  enableOperatorWebhook: false
  # Disable drain is needed for single-node OpenShift.
  disableDrain: true
  logLevel: 0
```

表6.5 シングルノード OpenShift クラスター用の SriovOperatorConfig CR オプション

SriovOperatorConfig CR フィールド	説明

spec.enableInjector	**Injector** Pod を無効にして、管理 Pod の数を減らします。**Injector** Pod を有効にした状態で開始し、必ずユーザーマニフェストを確認した後に無効にします。インジェクターが無効になっている場合、SR-IOV リソースを使用するコンテナーは、コンテナー仕様の **requests** および **limits** セクションで SR-IOV リソースを明示的に割り当てる必要があります。
	
	以下に例を示します。
	
	`containers:`
	`- name: my-sriov-workload-container`
	`  resources:`
	`    limits:`
	`      openshift.io/<resource_name>:  "1"`
	`    requests:`
	`      openshift.io/<resource_name>:  "1"`

spec.enableOperatorWebhook	**OperatorWebhook** Pod を無効にして、管理 Pod の数を減らします。**OperatorWebhook** Pod を有効にした状態で開始し、必ずユーザーマニフェストを確認した後に無効にします。

推奨される SriovNetwork 設定 (SriovNetwork.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
```

```
metadata:
  name: ""
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  #  resourceName: ""
  networkNamespace: openshift-sriov-network-operator
#  vlan: ""
#  spoofChk: ""
#  ipam: ""
#  linkState: ""
#  maxTxRate: ""
#  minTxRate: ""
#  vlanQoS: ""
#  trust: ""
#  capabilities: ""
```

表6.6 シングルノード OpenShift クラスター用の SriovNetwork CR オプション

SriovNetwork CR フィールド	説明
spec.vlan	**vlan** をミッドホールネットワーク用の VLAN で設定します。

推奨される SriovNetworkNodePolicy CR 設定 (SriovNetworkNodePolicy.yaml)

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: $name
  namespace: openshift-sriov-network-operator
  annotations: {}
spec:
  # The attributes for Mellanox/Intel based NICs as below.
  #     deviceType: netdevice/vfio-pci
  #     isRdma: true/false
  deviceType: $deviceType
  isRdma: $isRdma
  nicSelector:
    # The exact physical function name must match the hardware used
    pfNames: [$pfNames]
  nodeSelector:
    node-role.kubernetes.io/$mcp: ""
  numVfs: $numVfs
  priority: $priority
  resourceName: $resourceName
```

表6.7 シングルノード OpenShift クラスター用の SriovNetworkPolicy CR オプション

SriovNetworkNodePolicy CR フィールド	説明

SriovNetworkNodePolicy CR フィールド	説明
spec.deviceType	**deviceType** を、**vfio-pci** または **netdevice** として設定します。Mellanox NIC の場合は、**deviceType: netdevice** と **isRdma: true** を設定します。Intel ベースの NIC の場合は、**deviceType: vfio-pci** と **isRdma: false** を設定します。
spec.nicSelector.pfNames	フロントホールネットワークに接続されているインターフェイスを指定します。
spec.numVfs	フロントホールネットワークの VF の数を指定します。
spec.nicSelector.pfNames	物理機能の正確な名前は、ハードウェアと一致する必要があります。

推奨される SR-IOV カーネル設定 (07-sriov-relative-kernel-args-master.yaml)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 07-sriov-related-kernel-args-master
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
    - intel_iommu=on
    - iommu=pt
```

6.7.9. Console Operator

クラスターケイパビリティー機能を使用して、コンソールオペレーターがインストールされないようにします。ノードが一元的に管理されている場合は必要ありません。Operator を削除すると、アプリケーションのワークロードに追加の領域と容量ができます。

マネージドクラスターのインストール中に Console Operator を無効にするには、**SiteConfig** カスタムリソース (CR) の **spec.clusters.0.installConfigOverrides** フィールドで次のように設定します。

```
installConfigOverrides:  "{\"capabilities\":{\"baselineCapabilitySet\": \"None\" }}"
```

6.7.10. Alertmanager

DU ワークロードを実行する単一ノードの OpenShift クラスターでは、OpenShift Container Platform モニタリングコンポーネントによって消費される CPU リソースを削減する必要があります。以下の **ConfigMap** カスタムリソース (CR) は Alertmanager を無効にします。

推奨されるクラスターモニタリング設定 (ReduceMonitoringFootprint.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
  annotations: {}
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
    telemeterClient:
      enabled: false
    prometheusK8s:
       retention: 24h
```

6.7.11. Operator Lifecycle Manager

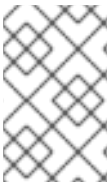
分散ユニットワークロードを実行するシングルノード OpenShift クラスターには、CPU リソースへの一貫したアクセスが必要です。Operator Lifecycle Manager (OLM) は定期的に Operator からパフォーマンスデータを収集するため、CPU 使用率が増加します。次の **ConfigMap** カスタムリソース (CR) は、OLM によるオペレーターパフォーマンスデータの収集を無効にします。

推奨されるクラスター OLM 設定 (ReduceOLMFootprint.yaml)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: collect-profiles-config
  namespace: openshift-operator-lifecycle-manager
data:
  pprof-config.yaml: |
    disabled: True
```

6.7.12. LVM Storage

論理ボリュームマネージャー (LVM) ストレージを使用して、シングルノード OpenShift クラスター上にローカルストレージを動的にプロビジョニングできます。



注記

シングルノード OpenShift の推奨ストレージソリューションは、Local Storage Operator です。LVM ストレージも使用できますが、その場合は追加の CPU リソースを割り当てる必要があります。

次の YAML の例では、OpenShift Container Platform アプリケーションで使用できるようにノードのストレージを設定しています。

推奨される LVMCluster 設定 (StorageLVMCluster.yaml)

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
```

```
metadata:
  name: odf-lvmcluster
  namespace: openshift-storage
spec:
  storage:
    deviceClasses:
    - name: vg1
      deviceSelector:
        paths:
        - /usr/disk/by-path/pci-0000:11:00.0-nvme-1
      thinPoolConfig:
        name: thin-pool-1
        overprovisionRatio: 10
        sizePercent: 90
```

表6.8 シングルノード OpenShift クラスター用の LVMCluster CR オプション

LVMCluster CR フィールド	説明
deviceSelector.paths	LVM ストレージに使用されるディスクを設定します。ディスクが指定されていない場合、LVM ストレージは指定されたシンプール内のすべての未使用ディスクを使用します。

6.7.13. ネットワーク診断

DU ワークロードを実行する単一ノードの OpenShift クラスターでは、これらの Pod によって作成される追加の負荷を軽減するために、Pod 間のネットワーク接続チェックが少なくて済みます。次のカスタムリソース (CR) は、これらのチェックを無効にします。

推奨されるネットワーク診断設定 (DisableSnoNetworkDiag.yaml)

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
  annotations: {}
spec:
  disableNetworkDiagnostics: true
```

関連情報

- ZTP を使用した遠端サイトのデプロイメント

第7章 VDU アプリケーションワークロードの単一ノード OPENSHIFT クラスターチューニングの検証

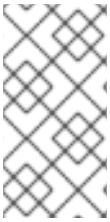
仮想化分散ユニット (vDU) アプリケーションをデプロイする前に、クラスターホストファームウェアおよびその他のさまざまなクラスター設定を調整および設定する必要があります。以下の情報を使用して、vDU ワークロードをサポートするためのクラスター設定を検証します。

関連情報

- GitOps ZTP を使用した単一ノードの OpenShift でのワークロードの分割
- シングルノード OpenShift に vDU をデプロイするためのリファレンス設定

7.1. VDU クラスターホストの推奨ファームウェア設定

OpenShift Container Platform 4.15 で実行される vDU アプリケーションのクラスターホストファームウェアを設定するための基礎として、以下の表を使用してください。



注記

次の表は、vDU クラスターホストファームウェア設定の一般的な推奨事項です。正確なファームウェア設定は、要件と特定のハードウェアプラットフォームによって異なります。ファームウェアの自動設定は、ゼロタッチプロビジョニングパイプラインでは処理されません。

表7.1 推奨されるクラスターホストファームウェア設定

ファームウェア設定	設定	説明
HyperTransport (HT)	有効	HyperTransport (HT) バスは、AMD が開発したバス技術です。HT は、ホストメモリー内のコンポーネントと他のシステムペリフェラル間の高速リンクを提供します。
UEFI	有効	vDU ホストの UEFI からの起動を有効にします。
CPU パワーとパフォーマンスポリシー	パフォーマンス	CPU パワーとパフォーマンスポリシーを設定し、エネルギー効率よりもパフォーマンスを優先してシステムを最適化します。
Uncore Frequency Scaling	Disabled	Uncore Frequency Scaling を無効にして、CPU のコア以外の部分の電圧と周波数が個別に設定されるのを防ぎます。
Uncore Frequency	最大	キャッシュやメモリーコントローラーなど、CPU のコア以外の部分を可能な最大動作周波数に設定します。
パフォーマンスの制限	Disabled	プロセッサーの Uncore Frequency 調整を防ぐために、パフォーマンス P 制限を無効にします。

ファームウェア設定	設定	説明
強化された Intel® SpeedStep テクノロジー	有効	Enhanced Intel SpeedStep を有効にして、システムがプロセッサーの電圧とコア周波数を動的に調整できるようにし、ホストの消費電力と発熱を減らします。
Intel® Turbo Boost Technology	有効	Intel ベースの CPU で Turbo Boost Technology を有効にすると、プロセッサーコアが電力、電流、および温度の仕様制限を下回って動作している場合、自動的に定格動作周波数よりも高速に動作できるようにします。
Intel Configurable TDP	有効	CPU の Thermal Design Power (TDP) を有効にします。
設定可能な TDP レベル	レベル 2	TDP レベルは、特定のパフォーマンス評価に必要な CPU 消費電力を設定します。TDP レベル 2 は、消費電力を犠牲にして、CPU を最も安定したパフォーマンスレベルに設定します。
energy Efficient Turbo	Disabled	Energy Efficient Turbo を無効にして、プロセッサーがエネルギー効率ベースのポリシーを使用しないようにします。
Hardware P-States	有効化または無効化	OS 制御の P-States を有効にして、省電力設定を許可します。**P-states** (パフォーマンスステート) を無効にして、オペレーティングシステムと CPU を最適化し、電力消費に対するパフォーマンスを向上させます。
Package C-State	C0/C1 の状態	C0 または C1 状態を使用して、プロセッサーを完全にアクティブな状態 (C0) に設定するか、ソフトウェアで実行されている CPU 内部クロックを停止します (C1)。
C1E	Disabled	CPU Enhanced Halt (C1E) は、Intel チップの省電力機能です。C1E を無効にすると、非アクティブ時にオペレーティングシステムが停止コマンドを CPU に送信することを防ぎます。
Processor C6	Disabled	C6 節電は、アイドル状態の CPU コアとキャッシュを自動的に無効にする CPU 機能です。C6 を無効にすると、システムパフォーマンスが向上します。
サブ NUMA クラスタリング	Disabled	サブ NUMA クラスタリングは、プロセッサーコア、キャッシュ、およびメモリーを複数の NUMA ドメインに分割します。このオプションを無効にすると、レイテンシーの影響を受けやすいワークロードのパフォーマンスが向上します。

注記

ホストのファームウェアでグローバル SR-IOV および VT-d 設定を有効にします。これらの設定は、ベアメタル環境に関連します。

注記

C-states と OS 制御の **P-States** の両方を有効にして、Pod ごとの電源管理を許可します。

7.2. VDU アプリケーションを実行するための推奨クラスター設定

仮想化分散ユニット (vDU) アプリケーションを実行するクラスターには、高度に調整かつ最適化された設定が必要です。以下では、OpenShift Container Platform 4.15 クラスターで vDU ワークロードをサポートするために必要なさまざまな要素について説明します。

7.2.1. シングルノード OpenShift クラスター用の推奨クラスター MachineConfig CR

ztp-site-generate コンテナーから抽出した **MachineConfig** カスタムリソース (CR) がクラスターに適用されていることを確認します。CR は、抽出した **out/source-crs/extra-manifest/** フォルダーにあります。

ztp-site-generate コンテナーからの次の **MachineConfig** CR は、クラスターホストを設定します。

表7.2 推奨される GitOps ZTP MachineConfig CR

MachineConfig CR	説明
01-container-mount-ns-and-kubelet-conf-master.yaml **01-container-mount-ns-and-kubelet-conf-worker.yaml**	コンテナーマウント namespace と Kubelet 設定を設定します。
03-sctp-machine-config-master.yaml **03-sctp-machine-config-worker.yaml**	SCTP カーネルモジュールをロードします。これらの **MachineConfig** CR は任意であり、このカーネルモジュールが必要ない場合は省略できます。
05-kdump-config-master.yaml **05-kdump-config-worker.yaml** **06-kdump-master.yaml** **06-kdump-worker.yaml**	クラスターの kdump クラッシュレポートを設定します。
07-sriov-related-kernel-args-master.yaml	クラスターの SR-IOV カーネル引数を設定します。
08-set-rcu-normal-master.yaml **08-set-rcu-normal-worker.yaml**	クラスターの再起動後に **rcu_expedited** モードを無効にします。
99-crio-disable-wipe-master.yaml **99-crio-disable-wipe-worker.yaml**	クラスター再起動後の自動 CRI-O キャッシュワイプを無効にします。

MachineConfig CR	説明
99-sync-time-once-master.yaml **99-sync-time-once-worker.yaml**	Chrony サービスによるシステムクロックのワンタイムチェックと調整を設定します。
enable-crun-master.yaml **enable-crun-worker.yaml**	**crun** OCI コンテナーランタイムを有効にします。
extra-manifest/enable-cgroups-v1.yaml **source-crs/extra-manifest/enable-cgroups-v1.yaml**	クラスターのインストール時および RHACM クラスターポリシーの生成時に cgroups v1 を有効にします。

注記

OpenShift Container Platform 4.14 以降では、**SiteConfig** CR の **cpuPartitioningMode** フィールドを使用してワークロードの分割を設定します。

関連情報

- GitOps ZTP を使用した単一ノードの OpenShift でのワークロードの分割
- ztp-site-generate コンテナーからのソース CR の抽出

7.2.2. 推奨されるクラスター Operator

次の Operator は、仮想化分散ユニット (vDU) アプリケーションを実行するクラスターに必要であり、ベースライン参照設定の一部です。

- Node Tuning Operator (NTO)。NTO は、以前は Performance Addon Operator で提供されていた機能をパッケージ化し、現在は NTO の一部になっています。
- PTP Operator
- SR-IOV Network Operator
- Red Hat OpenShift Logging Operator
- Local Storage Operator

7.2.3. 推奨されるクラスターカーネル設定

クラスターでは常に、サポートされている最新のリアルタイムカーネルバージョンを使用してください。クラスターに次の設定を適用していることを確認します。

1. 次の **additionalKernelArgs** がクラスターパフォーマンスプロファイルに設定されていることを確認します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
# ...
```

```
spec:
  additionalKernelArgs:
  - "rcupdate.rcu_normal_after_boot=0"
  - "efi=runtime"
  - "vfio_pci.enable_sriov=1"
  - "vfio_pci.disable_idle_d3=1"
  - "module_blacklist=irdma"

  # ...
```

2. オプション： **hardwareTuning** フィールドで CPU 周波数を設定します。
ハードウェアチューニングを使用して、予約済みコアおよび分離されたコア CPU 用の CPU 周波数を調整できます。アプリケーションのような FlexRAN の場合、ハードウェアベンダーは、デフォルトの提供されている周波数を下回って CPU 周波数を実行することを推奨しています。周波数を設定する前に、プロセッサーの生成における最大周波数設定はハードウェアベンダーのガイドラインを参照することを強く推奨します。以下の例は、Sapphire Rapid ハードウェア用の予約済みおよび分離された CPU の周波数を示しています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
      cpu:
        isolated: "2-19,22-39"
        reserved: "0-1,20-21"
      hugepages:
        defaultHugepagesSize: 1G
        pages:
          - size: 1G
            count: 32
      realTimeKernel:
          enabled: true
      hardwareTuning:
          isolatedCpuFreq: 2500000
          reservedCpuFreq: 2800000
```

3. **Tuned** CR の **performance-patch** プロファイルが、関連する **PerformanceProfile** CR の **isolated** CPU セットと一致する正しい CPU 分離セットを設定していることを確認します。次に例を示します。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: performance-patch
      # The 'include' line must match the associated PerformanceProfile name, for example:
      # include=openshift-node-performance-${PerformanceProfile.metadata.name}
      # When using the standard (non-realtime) kernel, remove the kernel.timer_migration
override from the [sysctl] section
```

```
    data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
      [scheduler]
      group.ice-ptp=0:f:10:*:ice-ptp.*
      group.ice-gnss=0:f:10:*:ice-gnss.*
      group.ice-dplls=0:f:10:*:ice-dplls.*
      [service]
      service.stalld=start,enable
      service.chronyd=stop,disable
# ...
```

7.2.4. リアルタイムカーネルバージョンの確認

OpenShift Container Platform クラスターでは常にリアルタイムカーネルの最新バージョンを使用してください。クラスターで使用されているカーネルバージョンが不明な場合は、次の手順で現在のリアルタイムカーネルバージョンとリリースバージョンを比較できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- **podman** 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、クラスターのバージョンを取得します。

```
$ OCP_VERSION=$(oc get clusterversion version -o jsonpath='{.status.desired.version}
{"\n"}')
```

2. リリースイメージの SHA 番号を取得します。

```
$ DTK_IMAGE=$(oc adm release info --image-for=driver-toolkit quay.io/openshift-release-
dev/ocp-release:$OCP_VERSION-x86_64)
```

3. リリースイメージコンテナーを実行し、クラスターの現在のリリースにパッケージ化されているカーネルバージョンを抽出します。

```
$ podman run --rm $DTK_IMAGE rpm -qa | grep 'kernel-rt-core-' | sed 's#kernel-rt-core-##'
```

出力例

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

これは、リリースに同梱されているデフォルトのリアルタイムカーネルバージョンです。

注記

リアルタイムカーネルは、カーネルバージョンの文字列 **.rt** で示されます。

検証

クラスターの現在のリリース用にリストされているカーネルバージョンが、クラスターで実行されている実際のリアルタイムカーネルと一致することを確認します。次のコマンドを実行して、実行中のリアルタイムカーネルバージョンを確認します。

1. クラスターノードへのリモートシェル接続を開きます。

```
$ oc debug node/<node_name>
```

2. リアルタイムカーネルバージョンを確認します。

```
sh-4.4# uname -r
```

出力例

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

7.3. 推奨されるクラスター設定が適用されていることの確認

クラスターが正しい設定で実行されていることを確認できます。以下の手順では、DU アプリケーションを OpenShift Container Platform 4.15 クラスターにデプロイするために必要なさまざまな設定を確認する方法について説明します。

前提条件

- クラスターをデプロイし、vDU ワークロード用に調整している。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. デフォルトの Operator Hub ソースが無効になっていることを確認します。以下のコマンドを実行します。

```
$ oc get operatorhub cluster -o yaml
```

出力例

```
spec:
  disableAllDefaultSources: true
```

2. 次のコマンドを実行して、必要なすべての **CatalogSource** リソースにワークロードのパーティショニング (**PreferredDuringScheduling**) のアノテーションが付けられていることを確認します。

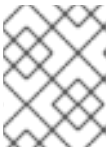
```
$ oc get catalogsource -A -o jsonpath='{range .items[*]}{.metadata.name}{" -- "}
{.metadata.annotations.target\.workload\.openshift\.io/management}{"\n"}{end}'
```

出力例

```
-
```

```
certified-operators -- {"effect": "PreferredDuringScheduling"}
community-operators -- {"effect": "PreferredDuringScheduling"}
ran-operators 1
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

1 アノテーションが付けられていない **CatalogSource** リソースも返されます。この例では、**ran-operators CatalogSource** リソースにはアノテーションが付けられておらず、**PreferredDuringScheduling** アノテーションがありません。



注記

適切に設定された vDU クラスターでは、単一のアノテーション付きカタログソースのみがリスト表示されます。

3. 該当するすべての OpenShift Container Platform Operator の namespace がワークロードのパーティショニング用にアノテーションされていることを確認します。これには、コア OpenShift Container Platform とともにインストールされたすべての Operator と、参照 DU チューニング設定に含まれる追加の Operator のセットが含まれます。以下のコマンドを実行します。

```
$ oc get namespaces -A -o jsonpath='{range .items[*]}{.metadata.name}{" -- "}
{.metadata.annotations.workload\.openshift\.io/allowed}{"\n"}{end}'
```

出力例

```
default --
openshift-apiserver -- management
openshift-apiserver-operator -- management
openshift-authentication -- management
openshift-authentication-operator -- management
```

重要

追加の Operator は、ワークロードパーティショニングのためにアノテーションを付けてはなりません。前のコマンドからの出力では、追加の Operator が **--** セパレーターの右側に値なしでリストされている必要があります。

4. **ClusterLogging** 設定が正しいことを確認してください。以下のコマンドを実行します。

   a. 適切な入力ログと出力ログが設定されていることを確認します。

```
$ oc get -n openshift-logging ClusterLogForwarder instance -o yaml
```

出力例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  creationTimestamp: "2022-07-19T21:51:41Z"
  generation: 1
```

```
  name: instance
  namespace: openshift-logging
  resourceVersion: "1030342"
  uid: 8c1a842d-80c5-447a-9150-40350bdf40f0
spec:
  inputs:
  - infrastructure: {}
    name: infra-logs
  outputs:
  - name: kafka-open
    type: kafka
    url: tcp://10.46.55.190:9092/test
  pipelines:
  - inputRefs:
    - audit
    name: audit-logs
    outputRefs:
    - kafka-open
  - inputRefs:
    - infrastructure
    name: infrastructure-logs
    outputRefs:
    - kafka-open
...
```

b. キュレーションスケジュールがアプリケーションに適していることを確認します。

```
$ oc get -n openshift-logging clusterloggings.logging.openshift.io instance -o yaml
```

出力例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  creationTimestamp: "2022-07-07T18:22:56Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "235796"
  uid: ef67b9b8-0e65-4a10-88ff-ec06922ea796
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    curator:
      schedule: 30 3 * * *
    type: curator
  managementState: Managed
...
```

5. 次のコマンドを実行して、Web コンソールが無効になっている (**managementState: Removed**) ことを確認します。

```
$ oc get consoles.operator.openshift.io cluster -o jsonpath="{ .spec.managementState }"
```

出力例

```
Removed
```

6. 次のコマンドを実行して、クラスターノードで **chronyd** が無効になっていることを確認します。

```
$ oc debug node/<node_name>
```

ノードで **chronyd** のステータスを確認します。

```
sh-4.4# chroot /host
```

```
sh-4.4# systemctl status chronyd
```

出力例

```
● chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset:
enabled)
   Active: inactive (dead)
     Docs: man:chronyd(8)
           man:chrony.conf(5)
```

7. **linuxptp-daemon** コンテナーへのリモートシェル接続と PTP Management Client (**pmc**) ツールを使用して、PTP インターフェイスがプライマリークロックに正常に同期されていることを確認します。

   a. 次のコマンドを実行して、**$PTP_POD_NAME** 変数に **linuxptp-daemon** Pod の名前を設定します。

   ```
   $ PTP_POD_NAME=$(oc get pods -n openshift-ptp -l app=linuxptp-daemon -o name)
   ```

   b. 次のコマンドを実行して、PTP デバイスの同期ステータスを確認します。

   ```
   $ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
   /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
   ```

   出力例

   ```
   sending: GET PORT_DATA_SET
     3cecef.fffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
       portIdentity            3cecef.fffe.7a7020-1
       portState               SLAVE
       logMinDelayReqInterval  -4
       peerMeanPathDelay       0
       logAnnounceInterval     1
       announceReceiptTimeout  3
       logSyncInterval         0
       delayMechanism          1
   ```

```
    logMinPdelayReqInterval 0
    versionNumber           2
  3cecef.fffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-2
    portState               LISTENING
    logMinDelayReqInterval  0
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
```

c. 次の **pmc** コマンドを実行して、PTP クロックのステータスを確認します。

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET TIME_STATUS_NP'
```

出力例

```
sending: GET TIME_STATUS_NP
  3cecef.fffe.7a7020-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset              10 (1)
    ingress_time               1657275432697400530
    cumulativeScaledRateOffset +0.000000000
    scaledLastGmPhaseChange    0
    gmTimeBaseIndicator        0
    lastGmPhaseChange          0x0000'0000000000000000.0000
    gmPresent                  true (2)
    gmIdentity                 3c2c30.ffff.670e00
```

(1) **master_offset** は -100 から 100 ns の間である必要があります。

(2) PTP クロックがマスターに同期されており、ローカルクロックがグランドマスタークロックではないことを示します。

d. **/var/run/ptp4l.0.config** の値に対応する予期される **master offset** 値が **linuxptp-daemon-container** ログにあることを確認します。

```
$ oc logs $PTP_POD_NAME -n openshift-ptp -c linuxptp-daemon-container
```

出力例

```
phc2sys[56020.341]: [ptp4l.1.config] CLOCK_REALTIME phc offset  -1731092 s2 freq -
1546242 delay    497
ptp4l[56020.390]: [ptp4l.1.config] master offset         -2 s2 freq   -5863 path delay       541
ptp4l[56020.390]: [ptp4l.0.config] master offset         -8 s2 freq  -10699 path delay       533
```

8. 次のコマンドを実行して、SR-IOV 設定が正しいことを確認します。

a. **SriovOperatorConfig** リソースの **disableDrain** 値が **true** に設定されていることを確認します。

```
$ oc get sriovoperatorconfig -n openshift-sriov-network-operator default -o jsonpath="
{.spec.disableDrain}{'\n'}"
```

出力例

```
true
```

b. 次のコマンドを実行して、**SriovNetworkNodeState** 同期ステータスが **Succeeded** であることを確認します。

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o jsonpath="
{.items[*].status.syncStatus}{'\n'}"
```

出力例

```
Succeeded
```

c. SR-IOV 用に設定された各インターフェイスの下の仮想機能 (**Vfs**) の予想される数と設定が、**.status.interfaces** フィールドに存在し、正しいことを確認します。以下に例を示します。

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: sriovnetwork.openshift.io/v1
  kind: SriovNetworkNodeState
...
  status:
    interfaces:
    ...
    - Vfs:
      - deviceID: 154c
        driver: vfio-pci
        pciAddress: 0000:3b:0a.0
        vendor: "8086"
        vfID: 0
      - deviceID: 154c
        driver: vfio-pci
        pciAddress: 0000:3b:0a.1
        vendor: "8086"
        vfID: 1
      - deviceID: 154c
        driver: vfio-pci
        pciAddress: 0000:3b:0a.2
        vendor: "8086"
        vfID: 2
      - deviceID: 154c
        driver: vfio-pci
        pciAddress: 0000:3b:0a.3
        vendor: "8086"
```

```
      vfID: 3
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.4
      vendor: "8086"
      vfID: 4
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.5
      vendor: "8086"
      vfID: 5
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.6
      vendor: "8086"
      vfID: 6
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.7
      vendor: "8086"
      vfID: 7
```

9. クラスターパフォーマンスプロファイルが正しいことを確認します。**cpu** セクションと **hugepages** セクションは、ハードウェア設定によって異なります。以下のコマンドを実行します。

```
$ oc get PerformanceProfile openshift-node-performance-profile -o yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  creationTimestamp: "2022-07-19T21:51:31Z"
  finalizers:
  - foreground-deletion
  generation: 1
  name: openshift-node-performance-profile
  resourceVersion: "33558"
  uid: 217958c0-9122-4c62-9d4d-fdc27c31118c
spec:
  additionalKernelArgs:
  - idle=poll
  - rcupdate.rcu_normal_after_boot=0
  - efi=runtime
  cpu:
    isolated: 2-51,54-103
    reserved: 0-1,52-53
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 32
      size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
```

```
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
status:
  conditions:
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "True"
    type: Available
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "True"
    type: Upgradeable
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "False"
    type: Progressing
  - lastHeartbeatTime: "2022-07-19T21:51:31Z"
    lastTransitionTime: "2022-07-19T21:51:31Z"
    status: "False"
    type: Degraded
  runtimeClass: performance-openshift-node-performance-profile
  tuned: openshift-cluster-node-tuning-operator/openshift-node-performance-openshift-node-
performance-profile
```

注記

CPU 設定は、サーバーで使用可能なコアの数に依存し、ワークロードパーティショニングの設定に合わせる必要があります。**hugepages** の設定は、サーバーとアプリケーションに依存します。

10. 次のコマンドを実行して、**PerformanceProfile** がクラスターに正常に適用されたことを確認します。

```
$ oc get performanceprofile openshift-node-performance-profile -o jsonpath="{range
.status.conditions[*]}{ @.type }{' -- '}{@.status}{'\n'}{end}"
```

出力例

```
Available -- True
Upgradeable -- True
Progressing -- False
Degraded -- False
```

11. 次のコマンドを実行して、**Tuned** パフォーマンスパッチの設定を確認します。

```
$ oc get tuneds.tuned.openshift.io -n openshift-cluster-node-tuning-operator performance-
patch -o yaml
```

出力例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  creationTimestamp: "2022-07-18T10:33:52Z"
  generation: 1
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  resourceVersion: "34024"
  uid: f9799811-f744-4179-bf00-32d4436c08fd
spec:
  profile:
  - data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
      [bootloader]
      cmdline_crash=nohz_full=2-23,26-47 (1)
      [sysctl]
      kernel.timer_migration=1
      [scheduler]
      group.ice-ptp=0:f:10:*:ice-ptp.*
      [service]
      service.stalld=start,enable
      service.chronyd=stop,disable
    name: performance-patch
  recommend:
  - machineConfigLabels:
      machineconfiguration.openshift.io/role: master
    priority: 19
    profile: performance-patch
```

(1) **cmdline=nohz_full=** の cpu リストは、ハードウェア設定によって異なります。

12. 次のコマンドを実行して、クラスターネットワーク診断が無効になっていることを確認します。

```
$ oc get networks.operator.openshift.io cluster -o
jsonpath='{.spec.disableNetworkDiagnostics}'
```

出力例

```
true
```

13. **Kubelet** のハウスキーピング間隔が、遅い速度に調整されていることを確認します。これは、**containerMountNS** マシン設定で設定されます。以下のコマンドを実行します。

```
$ oc describe machineconfig container-mount-namespace-and-kubelet-conf-master | grep
OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
```

出力例

```
Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
```

14. 次のコマンドを実行して、Grafana と **alertManagerMain** が無効になっていること、および Prometheus の保持期間が 24 時間に設定されていることを確認します。

```
$ oc get configmap cluster-monitoring-config -n openshift-monitoring -o jsonpath="{
.data.config\.yaml }"
```

出力例

```
grafana:
  enabled: false
alertmanagerMain:
  enabled: false
prometheusK8s:
   retention: 24h
```

a. 次のコマンドを使用して、Grafana および **alertManagerMain** ルートがクラスター内に見つからないことを確認します。

```
$ oc get route -n openshift-monitoring alertmanager-main
```

```
$ oc get route -n openshift-monitoring grafana
```

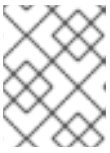
どちらのクエリーも **Error from server (NotFound)** メッセージを返す必要があります。

15. 次のコマンドを実行して、**PerformanceProfile**、**Tuned** performance-patch、ワークロードパーティショニング、およびカーネルコマンドライン引数のそれぞれに **reserved** として割り当てられた CPU が少なくとも 4 つあることを確認します。

```
$ oc get performanceprofile -o jsonpath="{ .items[0].spec.cpu.reserved }"
```

出力例

```
0-3
```



注記

ワークロードの要件によっては、追加の予約済み CPU の割り当てが必要になる場合があります。

第8章 SITECONFIG リソースを使用した高度なマネージドクラスター設定

SiteConfig カスタムリソース (CR) を使用して、インストール時にマネージドクラスターにカスタム機能と設定をデプロイできます。

8.1. GITOPS ZTP パイプラインでの追加インストールマニフェストのカスタマイズ

GitOps Zero Touch Provisioning (ZTP) パイプラインのインストールフェーズに追加するマニフェストセットを定義できます。これらのマニフェストは **SiteConfig** カスタムリソース (CR) にリンクされ、インストール時にクラスターに適用されます。インストール時に **MachineConfig** CR を含めると、インストール作業が効率的になります。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. GitOps ZTP パイプラインがクラスターインストールのカスタマイズ使用する、追加のマニフェスト CR のセットを作成します。

2. カスタム **/siteconfig** ディレクトリーに、追加のマニフェスト用のサブディレクトリー **/custom-manifest** を作成します。以下の例は、**/custom-manifest** フォルダーを持つ **/siteconfig** のサンプルを示しています。

```
siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
├── extra-manifest/
└── custom-manifest
    └── 01-example-machine-config.yaml
```

注記

全体で使用されているサブディレクトリー名 **/custom-manifest** および **/extra-manifest** は、名前の例にすぎません。これらの名前を使用する必要はなく、これらのサブディレクトリーに名前を付ける方法に制限はありません。この例では、**/extra-manifest** は、**ztp-site-generate** コンテナーの **/extra-manifest** の内容を保存する Git サブディレクトリーを指します。

3. カスタムの追加マニフェスト CR を **siteconfig/custom-manifest** ディレクトリーに追加します。

4. **SiteConfig** CR で、**extraManifests.searchPaths** フィールドにディレクトリー名を入力します。例:

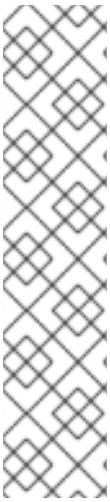
```
clusters:
- clusterName: "example-sno"
  networkType: "OVNKubernetes"
```

```
extraManifests:
  searchPaths:
    - extra-manifest/ 1
    - custom-manifest/ 2
```

1. **ztp-site-generate** コンテナーからコピーされたマニフェストのフォルダー。
2. カスタムマニフェストのフォルダー。

5. **SiteConfig**、**/extra-manifest**、および **/custom-manifest** CR を保存し、サイト設定リポジトリーにプッシュします。

クラスターのプロビジョニング中に、GitOps ZTP パイプラインは、**/custom-manifest** ディレクトリー内の CR を、**extra-manifest/** に保存されている追加マニフェストのデフォルトのセットに追加します。



注記

バージョン 4.14 以降、**extraManifestPath** には非推奨の警告が表示されます。

extraManifestPath は引き続きサポートされていますが、**extraManifests.searchPaths** を使用することを推奨します。**SiteConfig** ファイルで **extraManifests.searchPaths** を定義すると、GitOps ZTP パイプラインはサイトのインストール中に **ztp-site-generate** コンテナーからマニフェストを取得しません。

Siteconfig CR で **extraManifestPath** と **extraManifests.searchPaths** の両方を定義した場合は、**extraManifests.searchPaths** に定義された設定が優先されます。

/extra-manifest の内容を **ztp-site-generate** コンテナーから抽出し、GIT リポジトリーにプッシュすることを強く推奨します。

8.2. SITECONFIG フィルターを使用したカスタムリソースのフィルタリング

フィルターを使用すると、**SiteConfig** カスタムリソース (CR) を簡単にカスタマイズして、GitOps Zero Touch Provisioning (ZTP) パイプラインのインストールフェーズで使用する他の CR を追加または除外できます。

SiteConfig CR の **inclusionDefault** 値として **include** または **exclude** を指定し、さらに、含めたり除外したりする特定の **extraManifest** RAN CR のリストを指定することもできます。**inclusionDefault** を **include** に設定すると、GitOps ZTP パイプラインはインストール中に **/source-crs/extra-manifest** 内のすべてのファイルを適用します。**inclusionDefault** を **exclude** に設定すると、その逆になります。

デフォルトで含まれている **/source-crs/extra-manifest** フォルダーから個々の CR を除外できます。以下の例では、インストール時に **/source-crs/extra-manifest/03-sctp-machine-config-worker.yaml** CR を除外するようにカスタムの単一ノード OpenShift **SiteConfig** CR を設定します。

また、いくつかのオプションのフィルタリングシナリオも説明されています。

前提条件

- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。

- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. GitOps ZTP パイプラインが **03-sctp-machine-config-worker.yaml** CR ファイルを適用しないようにするには、**SiteConfig** CR で次の YAML を適用します。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "site1-sno-du"
  namespace: "site1-sno-du"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.16"
  sshPublicKey: "<ssh_public_key>"
  clusters:
- clusterName: "site1-sno-du"
  extraManifests:
    filter:
      exclude:
        - 03-sctp-machine-config-worker.yaml
```

   GitOps ZTP パイプラインは、インストール中に **03-sctp-machine-config-worker.yaml** CR をスキップします。**/source-crs/extra-manifest** 内の他のすべての CR が適用されます。

2. **SiteConfig** CR を保存し、変更をサイト設定リポジトリーにプッシュします。GitOps ZTP パイプラインは、**SiteConfig** フィルター命令に基づいて適用する CR を監視および調整します。

3. オプション: クラスターのインストール中に GitOps ZTP パイプラインがすべての **/source-crs/extra-manifest** CR を適用しないようにするには、**SiteConfig** CR で次の YAML を適用します。

```
- clusterName: "site1-sno-du"
  extraManifests:
    filter:
      inclusionDefault: exclude
```

4. オプション: インストール中にすべての **/source-crs/extra-manifest** RAN CR を除外し、代わりにカスタム CR ファイルを含めるには、カスタム **SiteConfig** CR を編集してカスタムマニフェストフォルダーと **include** ファイルを設定します。次に例を示します。

```
clusters:
- clusterName: "site1-sno-du"
  extraManifestPath: "<custom_manifest_folder>" 1
  extraManifests:
    filter:
      inclusionDefault: exclude 2
      include:
        - custom-sctp-machine-config-worker.yaml
```

1. **<custom_manifest_folder>** を、カスタムインストール CR を含むフォルダーの名前 (**user-custom-manifest/** など) に置き換えます。
2. インストール中に GitOps ZTP パイプラインが **/source-crs/extra-manifest** 内のファイルを適用しないようにするには、**inclusionDefault** を **exclude** に設定します。

次の例は、カスタムフォルダー構造を示しています。

```
siteconfig
  ├── site1-sno-du.yaml
  └── user-custom-manifest
      └── custom-sctp-machine-config-worker.yaml
```

8.3. SITECONFIG CR を使用してノードを削除する

SiteConfig カスタムリソース (CR) を使用すると、ノードを削除して再プロビジョニングできます。この方法は、手動でノードを削除するよりも効率的です。

前提条件

- 必要なインストールおよびポリシー CR を生成するようにハブクラスターを設定している。
- カスタムサイト設定データを管理できる Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. **SiteConfig** CR を更新して、**bmac.agent-install.openshift.io/remove-agent-and-node-on-delete=true** を追加します。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "cnfdf20"
  namespace: "cnfdf20"
spec:
  Clusters:
    nodes:
    - hostname: node6
      role: "worker"
      crAnnotations:
        add:
          BareMetalHost:
            bmac.agent-install.openshift.io/remove-agent-and-node-on-delete: true
# ...
```

2. **SiteConfig** CR を更新して **crSuppression.BareMetalHost** アノテーションを含めることで、**BareMetalHost** CR の生成を抑制します。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
```

```
  name: "cnfdf20"
  namespace: "cnfdf20"
spec:
  clusters:
  - nodes:
    - hostName: node6
      role: "worker"
      crSuppression:
      - BareMetalHost
# ...
```

3. 変更を Git リポジトリーにプッシュし、プロビジョニング解除が開始するまで待ちます。**BareMetalHost** CR のステータスが **deprovisioning** に変更されるはずです。**BareMetalHost** のプロビジョニング解除が完了し、完全に削除されるまで待ちます。

検証

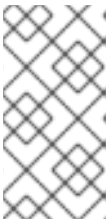
1. 次のコマンドを実行して、ワーカーノードの **BareMetalHost** および **Agent** CR がハブクラスターから削除されていることを確認します。

```
$ oc get bmh -n <cluster-ns>
```

```
$ oc get agent -n <cluster-ns>
```

2. 次のコマンドを実行して、スポーククラスターからノードレコードが削除されたことを確認します。

```
$ oc get nodes
```



注記

シークレットを操作している場合は、シークレットを削除するのが早すぎると、ArgoCD が削除後に再同期を完了するためにシークレットを必要とするため、問題が発生する可能性があります。現在の ArgoCD 同期が完了したら、ノードのクリーンアップ後にのみシークレットを削除します。

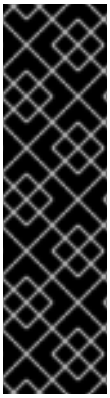
次のステップ

ノードを再プロビジョニングするには、以前に **SiteConfig** に追加された変更を削除し、変更を Git リポジトリーにプッシュして、同期が完了するまで待機します。これにより、ワーカーノードの **BareMetalHost** CR が再生成され、ノードの再インストールがトリガーされます。

第9章 POLICYGENERATOR リソースを使用したクラスターポリシーの管理

9.1. POLICYGENERATOR リソースを使用したマネージドクラスターポリシーの設定

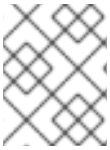
適用されたポリシーのカスタムリソース (CR) は、プロビジョニングするマネージドクラスターを設定します。Red Hat Advanced Cluster Management (RHACM) が PolicyGenTemplate CR を使用して、適用されるポリシー CR を生成する方法をカスタマイズできます。



重要

GitOps ZTP での PolicyGenerator リソースの使用は、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 製品サポート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場合があります。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。



注記

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

9.1.1. RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

PolicyGenerator カスタムリソース(CR)および **PolicyGenTemplate** CR を GitOps ZTP で使用して、マネージドクラスターの RHACM ポリシーを生成できます。

GitOps ZTP で OpenShift Container Platform リソースにパッチを適用する場合、**PolicyGenTemplate** CR を介して **PolicyGenerator** CR を使用する利点があります。RHACM **PolicyGenerator** API を使用すると、**PolicyGenTemplate** リソースでリソースにパッチを適用できない一般的な方法が提供されます。

PolicyGenerator API は Open Cluster Management 標準に含まれますが、**PolicyGenTemplate** API は含まれません。**PolicyGenerator** および **PolicyGenTemplate** リソースパッチおよび配置ストラテジーの比較は、以下の表で説明されています。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および **PolicyGenerator** CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

表9.1 RHACM PolicyGenerator および PolicyGenTemplate パッチ適用の比較

PolicyGenerator patching	PolicyGenTemplate のパッチ適用
リソースのマージに Kustomize 戦略的なマージを使用します。詳細は、Kustomize を使用した Kubernetes オブジェクトの宣言型管理 を参照してください。	変数をパッチで定義されている値に置き換えることで機能します。これは Kustomize マージストラテジーよりも柔軟性が低いです。
ManagedClusterSet および **Binding** リソースをサポートします。	**ManagedClusterSet** および **Binding** リソースはサポートされません。
パッチ適用にのみ依存します。埋め込み変数置換は必要ありません。	パッチで定義された変数値を上書きします。
マージパッチのマージリストをサポートしません。マージパッチのリストを置き換えることがサポートされています。	リストのマージと置換は、制限された方法でサポートされています。リストに 1 つのオブジェクトのみをマージできます。
現在、リソースのパッチ適用の OpenAPI 仕様 をサポートしていません。これは、スキーマ(**PtpConfig** リソースなど)に準拠していないコンテンツをマージするために、パッチに追加のディレクティブが必要であることを意味します。	フィールドと値を、パッチで定義されている値に置き換えることで機能します。
追加のディレクティブ（例： **$patch: は、スキーマに従わないコンテンツをマージするためにパッチ内で置き換える** ）が必要です。	ソース CR で定義されたフィールドと値を、パッチで定義された値（例： **$name** ）に置き換えます。
参照ソース CR で定義された **Name** フィールドおよび **Namespace** フィールドにパッチを適用できますが、CR ファイルに単一のオブジェクトがある場合のみです。	参照ソース CR で定義された **Name** フィールドおよび **Namespace** フィールドにパッチを適用できます。

9.1.2. PolicyGenerator CRD について

PolicyGenTemplate カスタムリソース定義 (CRD) は、 **PolicyGen** ポリシージェネレーターに、どのカスタムリソース (CR) をクラスター設定に含めるか、CR を生成されたポリシーに結合する方法、およびこれらの CR 内のどのアイテムをオーバーレイコンテンツで更新する必要があるかを伝えます。

次の例は、**ztp-site-generate** 参照コンテナーから抽出された **PolicyGenTemplate** CR (**common-du-ranGen.yaml**) を示しています。common-du-ranGen.yaml ファイルは、2 つの Red Hat Advanced Cluster Management (RHACM) ポリシーを定義します。ポリシーは、CR 内の **policyName** の一意の値

ごとに 1 つずつ、設定 CR のコレクションを管理します。**acm-common-du-ranGen.yaml** は、**policyDefaults.placement.labelSelector** セクションにリストされているラベルに基づいて、ポリシーをクラスターにバインドするための単一の配置バインディングと配置ルールを作成します。

Example PolicyGenerator CR – acm-common-ranGen.yaml

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
   name: common-latest
placementBindingDefaults:
   name: common-latest-placement-binding (1)
policyDefaults:
   namespace: ztp-common
   placement:
      labelSelector:
         matchExpressions:
            - key: common
              operator: In
              values:
                - "true"
            - key: du-profile
              operator: In
              values:
                - latest
   remediationAction: inform
   severity: low
   namespaceSelector:
      exclude:
         - kube-*
      include:
         - '*'
   evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
   - name: common-latest-config-policy
     policyAnnotations:
       ran.openshift.io/ztp-deploy-wave: "1"
     manifests:
       - path: source-crs/ReduceMonitoringFootprint.yaml
       - path: source-crs/DefaultCatsrc.yaml (2)
         patches:
           - metadata:
               name: redhat-operators-disconnected
             spec:
               displayName: disconnected-redhat-operators
               image: registry.example.com:5000/disconnected-redhat-operators/disconnected-redhat-
operator-index:v4.9
       - path: source-crs/DisconnectedICSP.yaml
         patches:
           - spec:
               repositoryDigestMirrors:
                  - mirrors:
                      - registry.example.com:5000
                    source: registry.redhat.io
```

```
  - name: common-latest-subscriptions-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests: 3
      - path: source-crs/SriovSubscriptionNS.yaml
      - path: source-crs/SriovSubscriptionOperGroup.yaml
      - path: source-crs/SriovSubscription.yaml
      - path: source-crs/SriovOperatorStatus.yaml
      - path: source-crs/PtpSubscriptionNS.yaml
      - path: source-crs/PtpSubscriptionOperGroup.yaml
      - path: source-crs/PtpSubscription.yaml
      - path: source-crs/PtpOperatorStatus.yaml
      - path: source-crs/ClusterLogNS.yaml
      - path: source-crs/ClusterLogOperGroup.yaml
      - path: source-crs/ClusterLogSubscription.yaml
      - path: source-crs/ClusterLogOperatorStatus.yaml
      - path: source-crs/StorageNS.yaml
      - path: source-crs/StorageOperGroup.yaml
      - path: source-crs/StorageSubscription.yaml
      - path: source-crs/StorageOperatorStatus.yaml
```

1 このラベルを持つすべてのクラスターにポリシーを適用します。

2 **DefaultCatsrc.yaml** ファイルには、切断されたレジストリーのカタログソースと関連するレジストリー設定の詳細が含まれます。

3 **policies.manifests** の下にリストされているファイルは、インストールされたクラスターの Operator ポリシーを作成します。

PolicyGenerator CR は、任意の数の組み込み CR で設定できます。次の例の CR をハブクラスターに適用して、単一の CR を含むポリシーを生成します。

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno
placementBindingDefaults:
  name: group-du-sno-placement-binding
policyDefaults:
  namespace: ztp-group
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
```

```
policies:
  - name: group-du-sno-config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: '10'
    manifests:
      - path: source-crs/PtpConfigSlave-MCP-master.yaml
        patches:
          - metadata: null
            name: du-ptp-slave
            namespace: openshift-ptp
            annotations:
              ran.openshift.io/ztp-deploy-wave: '10'
            spec:
              profile:
                - name: slave
                  interface: $interface
                  ptp4lOpts: '-2 -s'
                  phc2sysOpts: '-a -r -n 24'
                  ptpSchedulingPolicy: SCHED_FIFO
                  ptpSchedulingPriority: 10
                  ptpSettings:
                    logReduce: 'true'
                  ptp4lConf: |
                    [global]
                    #
                    # Default Data Set
                    #
                    twoStepFlag 1
                    slaveOnly 1
                    priority1 128
                    priority2 128
                    domainNumber 24
                    #utc_offset 37
                    clockClass 255
                    clockAccuracy 0xFE
                    offsetScaledLogVariance 0xFFFF
                    free_running 0
                    freq_est_interval 1
                    dscp_event 0
                    dscp_general 0
                    dataset_comparison G.8275.x
                    G.8275.defaultDS.localPriority 128
                    #
                    # Port Data Set
                    #
                    logAnnounceInterval -3
                    logSyncInterval -4
                    logMinDelayReqInterval -4
                    logMinPdelayReqInterval -4
                    announceReceiptTimeout 3
                    syncReceiptTimeout 0
                    delayAsymmetry 0
                    fault_reset_interval -4
                    neighborPropDelayThresh 20000000
                    masterOnly 0
                    G.8275.portDS.localPriority 128
```

```
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
```

```
                delay_filter_length 10
                egressLatency 0
                ingressLatency 0
                boundary_clock_jbod 0
                #
                # Clock description
                #
                productDescription ;;
                revisionData ;;
                manufacturerIdentity 00:00:00
                userDescription ;
                timeSource 0xA0
            recommend:
              - profile: slave
                priority: 4
                match:
                  - nodeLabel: node-role.kubernetes.io/master
```

ソースファイル **PtpConfigSlave.yaml** を例として使用すると、ファイルは **PtpConfig** CR を定義します。**PtpConfigSlave** サンプルの生成ポリシーは **group-du-sno-config-policy** という名前です。生成された **group-du-sno-config-policy** に定義される **PtpConfig** CR は **du-ptp-slave** という名前です。**PtpConfigSlave.yaml** で定義された **spec** は、**du-ptp-slave** の下に、ソースファイルで定義された他の **spec** 項目と共に配置されます。

次の例は、**group-du-sno-config-policy** CR を示しています。

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
    name: du-upgrade
placementBindingDefaults:
    name: du-upgrade-placement-binding
policyDefaults:
    namespace: ztp-group-du-sno
    placement:
        labelSelector:
            matchExpressions:
                - key: group-du-sno
                  operator: Exists
    remediationAction: inform
    severity: low
    namespaceSelector:
        exclude:
            - kube-*
        include:
            - '*'
    evaluationInterval:
        compliant: 10m
        noncompliant: 10s
policies:
    - name: du-upgrade-operator-catsrc-policy
      policyAnnotations:
        ran.openshift.io/ztp-deploy-wave: "1"
      manifests:
        - path: source-crs/DefaultCatsrc.yaml
```

```
patches:
  - metadata:
      name: redhat-operators
    spec:
      displayName: Red Hat Operators Catalog
      image: registry.example.com:5000/olm/redhat-operators:v4.14
      updateStrategy:
        registryPoll:
          interval: 1h
    status:
      connectionState:
        lastObservedState: READY
```

9.1.3. PolicyGenerator CR をカスタマイズする際の推奨事項

サイト設定の **PolicyGenTemplate** カスタムリソース (CR) をカスタマイズするときは、次のベストプラクティスを考慮してください。

- 必要な数のポリシーを使用します。使用するポリシーが少ないほど、必要なリソースが少なくなります。追加のポリシーごとに、ハブクラスターとデプロイされたマネージドクラスターの CPU 負荷が増大します。CR は **PolicyGenTemplate** CR の **policyName** フィールドに基づいてポリシーに統合されます。policyName に同じ値を持つ同じ PolicyGenTemplate の CR は単一のポリシーで管理されます。

- 切断された環境では、すべての Operator を含む単一のインデックスとしてレジストリーを設定することにより、すべての Operator に対して単一のカタログソースを使用します。マネージドクラスターに **CatalogSource** CR を追加するたびに、CPU 使用率が増加します。

- **MachineConfig** CR は、インストール時に適用されるように **SiteConfig** CR に **追加の Manifest** として組み込む必要があります。これにより、クラスターがアプリケーションをデプロイする準備ができるまで全体的な時間がかかる可能性があります。

- **PolicyGenerator** CR は、目的のバージョンを明示的に識別するために channel フィールドをオーバーライドする必要があります。これにより、アップグレード時にソース CR が変更されても、生成されたサブスクリプションが更新されないようになります。

関連情報

- RHACM を使用したクラスターのスケーリングに関する推奨事項は、パフォーマンスおよびスケーラビリティー を参照してください。

注記

ハブクラスターで多数のスポーククラスターを管理する場合は、ポリシーの数を最小限に抑えてリソースの消費を減らします。

複数のコンフィギュレーション CR を 1 つまたは限られた数のポリシーにグループ化することは、ハブクラスター上のポリシーの総数を減らすための 1 つの方法です。サイト設定の管理に共通、グループ、サイトというポリシーの階層を使用する場合は、サイト固有の設定を 1 つのポリシーにまとめることが特に重要である。

9.1.4. RAN デプロイメントの PolicyGenerator CR

PolicyGenTemplate (PGT) カスタムリソース (CR) を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してクラスターに適用される設定をカスタマイズします。PGT CR を使用す

ると、1つ以上のポリシーを生成して、クラスターのフリートで設定 CR のセットを管理できます。PGT は、管理された CR のセットを識別し、それらをポリシーにバンドルし、それらの CR をラップするポリシーを構築し、ラベルバインディングルールを使用してポリシーをクラスターに関連付けます。

GitOps ZTP コンテナーから取得した参照設定は、RAN (Radio Access Network) 分散ユニット (DU) アプリケーションに典型的な厳しいパフォーマンスとリソース利用制約をクラスターが確実にサポートできるように、重要な機能とノードのチューニング設定のセットを提供するように設計されています。ベースライン設定の変更または省略は、機能の可用性、パフォーマンス、およびリソースの利用に影響を与える可能性があります。参照 **PolicyGenTemplate** CR をベースに、お客様のサイト要件に合わせた設定ファイルの階層を作成します。

RAN DU クラスター設定に定義されているベースライン **PolicyGenTemplate** CR は、GitOps ZTP **ztp-site-generate** コンテナーから抽出することが可能です。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。

PolicyGenerator CR は、**./out/argocd/example/acmpolicygenerator/** フォルダーにあります。参照アーキテクチャーには、common、group、および site 固有の設定 CR があります。各 **PolicyGenerator** CR は **./out/source-crs** フォルダーにある他の CR を参照します。

RAN クラスター設定に関連する **PolicyGenerator** CR は以下で説明されています。バリアントは、単一ノード、3 ノードのコンパクト、および標準のクラスター設定の相違点に対応するために、グループ PolicyGenTemplate CR に提供されます。同様に、シングルノードクラスターとマルチノード (コンパクトまたはスタンダード) クラスターについても、サイト固有の設定バリエーションが提供されています。デプロイメントに関連するグループおよびサイト固有の設定バリアントを使用します。

表9.2 RAN デプロイメントの PolicyGenerator CR

PolicyGenerator CR	説明
acm-example-multinode-site.yaml	マルチノードクラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
acm-example-sno-site.yaml	単一ノードの OpenShift クラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
acm-common-mno-ranGen.yaml	マルチノードクラスターに適用される共通の RAN ポリシー設定のセットが含まれています。
acm-common-ranGen.yaml	すべてのクラスターに適用される共通の RAN CR のセットが含まれています。これらの CR は、RAN の典型的なクラスター機能とベースラインクラスターのチューニングを提供する Operator のセットをサブスクライブします。
acm-group-du-3node-ranGen.yaml	3 ノードクラスター用の RAN ポリシーのみが含まれています。
acm-group-du-sno-ranGen.yaml	シングルノードクラスター用の RAN ポリシーのみが含まれています。

PolicyGenerator CR	説明
acm-group-du-standard-ranGen.yaml	標準的な 3 つのコントロールプレーンクラスターの RAN ポリシーが含まれています。
acm-group-du-3node-validator-ranGen.yaml	**PolicyGenTemplate** CR は、3 ノードクラスターに必要なさまざまなポリシーを生成するために使用されます。
acm-group-du-standard-validator-ranGen.yaml	標準クラスターに必要なさまざまなポリシーを生成するために使用される **PolicyGenTemplate** CR。
acm-group-du-sno-validator-ranGen.yaml	**PolicyGenTemplate** CR は、単一ノードの OpenShift クラスターに必要なさまざまなポリシーを生成するために使用されます。

関連情報

- GitOps ZTP サイト設定リポジトリーの準備

9.1.5. PolicyGenerator CR を使用したマネージドクラスターのカスタマイズ

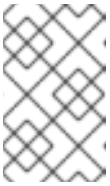
次の手順を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してプロビジョニングするマネージドクラスターに適用されるポリシーをカスタマイズします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. サイト固有の設定 CR の **PolicyGenerator** CR を作成します。
   a. CR の適切な例を **out/argocd/example/acmpolicygenerator/** フォルダーから選択します（例： **acm-example-sno-site.yaml** または **acm-example-multinode-site.yaml** ）。
   b. サンプルファイルの **policyDefaults.placement.labelSelector** フィールドを、**SiteConfig** CR に含まれるサイト固有のラベルと一致するように変更します。サンプルの **SiteConfig** ファイルでは、サイト固有のラベルは **sites: example-sno** です。



注記

PolicyGenerator policyDefaults.placement.labelSelector フィールドで定義されているラベルが、関連するマネージドクラスターの **SiteConfig** CR で定義されているラベルに対応していることを確認してください。

c. サンプルファイルの内容を目的の設定に合わせて変更します。

2. オプション: クラスターのフリート全体に適用される一般的な設定 CR の **PolicyGenTemplate** CR を作成します。

   a. **out/argocd/example/acmpolicygenerator/** フォルダーから CR の適切な例を選択します（例： **acm-common-ranGen.yaml** ）。

   b. サンプルファイルの内容を目的の設定に合わせて変更します。

3. オプション: フリート内のクラスターの特定のグループに適用されるグループ設定 CR の **PolicyGenTemplate** CR を作成します。
   オーバーレイド仕様ファイルの内容が必要な終了状態と一致することを確認します。
   out/source-crs ディレクトリーには、PolicyGenTemplate テンプレートに含めることができる source-crs の完全な一覧が含まれます。

注記

クラスターの特定の要件に応じて、クラスターの種類ごとに 1 つ以上のグループポリシーが必要になる場合があります。特に、サンプルのグループポリシーにはそれぞれ単一の **PerformancePolicy.yaml** ファイルがあり、それらのクラスターが同一のハードウェア設定である場合にのみクラスターのセット全体で共有できることを考慮しています。

   a. **out/argocd/example/acmpolicygenerator/** フォルダーから CR の適切な例を選択します（例： **acm-group-du-sno-ranGen.yaml** ）。

   b. サンプルファイルの内容を目的の設定に合わせて変更します。

4. オプション: GitOps ZTP のインストールとデプロイされたクラスターの設定が完了したときに通知するバリデータ通知ポリシー **PolicyGenTemplate** CR を作成します。詳細は、バリデータ通知ポリシーの作成を参照してください。

5. **out/argocd/example/acmpolicygenerator//ns.yaml** ファイルの例と同様の YAML ファイルで、すべてのポリシーの namespace を定義します。

重要

PolicyGenerator CR と同じファイルに **Namespace** CR を含めないでください。

6. **out/argocd/example/acmpolicygenerator/kustomization.yaml** に示されている例と同様に、**PolicyGenerator** CR と **Namespace** CR をジェネレーターセクションの **kustomization.yaml** ファイルに追加します。

7. **PolicyGenTemplate** CR、**Namespace** CR、および関連する **kustomization.yaml** ファイルを Git リポジトリーにコミットし、変更をプッシュします。
   ArgoCD パイプラインが変更を検出し、マネージドクラスターのデプロイを開始します。**SiteConfig** CR と **PolicyGenerator** CR に同時に変更をプッシュできます。

関連情報

- バリデーターインフォームポリシーを使用した GitOps ZTP クラスターデプロイメントの完了のシグナリング

9.1.6. マネージドクラスターポリシーのデプロイメントの進行状況の監視

ArgoCD パイプラインは、Git の **PolicyGenTemplate** CR を使用して RHACM ポリシーを生成し、ハブクラスターに同期します。支援されたサービスが OpenShift Container Platform をマネージドクラスターにインストールした後、管理対象クラスターのポリシー Synchronization の進行状況をモニターできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. Topology Aware Lifecycle Manager (TALM) は、クラスターにバインドされている設定ポリシーを適用します。
   クラスターのインストールが完了し、クラスターが **Ready** になると、**ran.openshift.io/ztp-deploy-wave** アノテーションで 定義された順序付きポリシーのリストで、このクラスターに対応する **ClusterGroupUpgrade** CR が TALM により自動的に作成されます。クラスターのポリシーは、**ClusterGroupUpgrade** CR に記載されている順序で適用されます。

   以下のコマンドを使用して、設定ポリシー調整のハイレベルの進捗を監視できます。

   ```
   $ export CLUSTER=<clusterName>
   ```

   ```
   $ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[-1:]}' | jq
   ```

   出力例

   ```
   {
     "lastTransitionTime": "2022-11-09T07:28:09Z",
     "message": "Remediating non-compliant policies",
     "reason": "InProgress",
     "status": "True",
     "type": "Progressing"
   }
   ```

2. RHACM ダッシュボードまたはコマンドラインを使用して、詳細なクラスターポリシーのコンプライアンスステータスを監視できます。

   a. **oc** を使用してポリシーのコンプライアンスを確認するには、次のコマンドを実行します。

      ```
      $ oc get policies -n $CLUSTER
      ```

      出力例

```
NAME                                          REMEDIATION ACTION   COMPLIANCE STATE
AGE
ztp-common.common-config-policy               inform               Compliant
3h42m
ztp-common.common-subscriptions-policy        inform               NonCompliant
3h42m
ztp-group.group-du-sno-config-policy          inform               NonCompliant
3h42m
ztp-group.group-du-sno-validator-du-policy    inform               NonCompliant
3h42m
ztp-install.example1-common-config-policy-pjz9s   enforce          Compliant
167m
ztp-install.example1-common-subscriptions-policy-zzd9k   enforce   NonCompliant
164m
ztp-site.example1-config-policy               inform               NonCompliant      3h42m
ztp-site.example1-perf-policy                 inform               NonCompliant      3h42m
```

b. RHACM Web コンソールからポリシーのステータスを確認するには、次のアクションを実行します。

   i. **ガバナンス → ポリシーの検索** をクリックします。

   ii. クラスターポリシーをクリックして、ステータスを確認します。

すべてのクラスターポリシーが準拠すると、クラスターの GitOps ZTP のインストールと設定が完了します。**ztp-done** ラベルがクラスターに追加されます。

参照設定では、準拠する最終的なポリシーは、***-du-validator-policy** ポリシーで定義されたものです。このポリシーは、クラスターに準拠する場合、すべてのクラスター設定、Operator のインストール、および Operator 設定が完了します。

9.1.7. 設定ポリシー CR の生成の検証

ポリシーのカスタムリソース (CR) は、作成元の **PolicyGenTemplate** と同じネームスペースで生成される。以下のコマンドを使用して示すように、ztp-common、**ztp-group**、または **ztp-site** ベースのいずれであるかにかかわらず、PolicyGenTemplate から生成されたすべてのポリシー CR に同じトラブルシューティングフローが適用されます。

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

予想される policy-wraped CR のセットが表示されるはずです。

ポリシーの同期に失敗した場合は、以下のトラブルシューティング手順を使用します。

手順

1. ポリシーの詳細情報を表示するには、次のコマンドを実行します。

   ```
   $ oc describe -n openshift-gitops application policies
   ```

2. **Status: Conditions:** の有無を確認し、エラーログを表示します。たとえば、無効な **sourceFile** エントリーを **fileName:** に設定すると、以下のようなエラーが発生します。

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or directory
Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit status 1: exit
status 1
    Type:  ComparisonError
```

3. **Status: Sync:** をチェックします。**Status: Conditions:**: でログエラーが発生した場合 **Status: Sync:** に **Unknown** または **Error** と表示されます。

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  policies-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:             policies
        Repo URL:         https://git.com/ran-sites/policies/.git
        Target Revision:  master
    Status:               Error
```

4. Red Hat Advanced Cluster Management (RHACM) が **ManagedCluster** オブジェクトにポリシーが適用されることを認識すると、ポリシー CR オブジェクトがクラスターネームスペースに適用されます。ポリシーがクラスターネームスペースにコピーされたかどうかを確認します。

```
$ oc get policy -n $CLUSTER
```

出力例:

```
NAME                                     REMEDIATION ACTION   COMPLIANCE STATE   AGE
ztp-common.common-config-policy          inform               Compliant          13d
ztp-common.common-subscriptions-policy   inform               Compliant          13d
ztp-group.group-du-sno-config-policy     inform               Compliant          13d
ztp-group.group-du-sno-validator-du-policy   inform           Compliant          13d
ztp-site.example-sno-config-policy       inform               Compliant          13d
```

RHACM は、適用可能なすべてのポリシーをクラスターの namespace にコピーします。コピーされたポリシー名の形式は < **PolicyGenerator.Namespace>.<PolicyGenerator.Name>-<policyName> です**。

5. クラスター namespace にコピーされないポリシーの配置ルールを確認します。これらのポリシーの **PlacementRule** の **matchSelector**、**ManagedCluster** オブジェクトのラベルと一致する必要があります。

```
$ oc get Placement -n $NS
```

6. **PlacementRule** 名は、以下のコマンドを使用して、不足しているポリシー (common、group、または site) に適した名前であることに注意してください。

```
$ oc get Placement -n $NS <placement_rule_name> -o yaml
```

- status-decisions にはクラスター名が含まれている必要があります。
- spec の **matchSelector** の key-value ペアは、マネージドクラスター上のラベルと一致する必要があります。

7. 以下のコマンドを使用して、**ManagedCluster** オブジェクトのラベルを確認します。

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. 以下のコマンドを使用して、準拠しているポリシーを確認します。

```
$ oc get policy -n $CLUSTER
```

Namespace、**OperatorGroup**、および **Subscription** ポリシーが準拠しているが Operator 設定ポリシーが該当しない場合、Operator はマネージドクラスターにインストールされていない可能性があります。このため、スポークに CRD がまだ適用されていないため、Operator 設定ポリシーの適用に失敗します。

9.1.8. ポリシー調整の再開

たとえば、**ClusterGroupUpgrade** カスタムリソース (CR) がタイムアウトした場合など、予期しないコンプライアンスの問題が発生した場合は、ポリシー調整を再開できます。

手順

1. **ClusterGroupUpgrade** CR は、管理クラスターの状態が **Ready** になった後に Topology Aware Lifecycle Manager によって namespace **ztp-install** に生成されます。

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. 予期せぬ問題が発生し、設定されたタイムアウト (デフォルトは 4 時間) 内にポリシーが苦情にならなかった場合、**ClusterGroupUpgrade** CR のステータスは **UpgradeTimedOut** と 表示されます。

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[?
(@.type=="Ready")]}'
```

3. **UpgradeTimedOut** 状態の **ClusterGroupUpgrade** CR は、1 時間ごとにポリシー照合を自動的に再開します。ポリシーを変更した場合は、既存の **ClusterGroupUpgrade** CR を削除して再試行をすぐに開始できます。これにより、ポリシーをすぐに調整する新規 **ClusterGroupUpgrade** CR の自動作成がトリガーされます。

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

ClusterGroupUpgrade CR が **UpgradeCompleted** のステータスで完了し、管理対象のクラスターに **ztp-done** ラベルが適用されると、**PolicyGenTemplate** を使用して追加の設定変更を行うことができます。既存の **ClusterGroupUpgrade** CR を削除しても、TALM は新規 CR を生成しません。

この時点で、GitOps ZTP はクラスターとの対話を完了しました。それ以降の対話は更新として扱われ、ポリシーの修復のために新しい **ClusterGroupUpgrade** CR が作成されます。

関連情報

- Topology Aware Lifecycle Manager (TALM) を使用して独自の **ClusterGroupUpgrade** CR を作成する方法は、ClusterGroupUpgrade CR について を参照してください。

9.1.9. ポリシーを使用して適用済みマネージドクラスター CR を変更する

ポリシーを使用して、マネージドクラスターにデプロイされたカスタムリソース (CR) からコンテンツを削除できます。

PolicyGenTemplate CR から作成されたすべての **Policy** CR は、**complianceType** フィールドがデフォルトで **musthave** に設定されています。マネージドクラスター上の CR には指定されたコンテンツがすべて含まれているため、コンテンツが削除されていない **musthave** ポリシーは依然として準拠しています。この設定では、CR からコンテンツを削除すると、TALM はポリシーからコンテンツを削除しますが、そのコンテンツはマネージドクラスター上の CR からは削除されません。

complianceType フィールドを **Mustonlyhave** に設定することで、ポリシーはクラスター上の CR がポリシーで指定されている内容と完全に一致するようにします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- RHACM を実行しているハブクラスターからマネージドクラスターをデプロイしている。
- ハブクラスターに Topology Aware Lifecycle Manager がインストールされている。

手順

1. 影響を受ける CR から不要になったコンテンツを削除します。この例では、**SriovOperatorConfig** CR から **disableDrain: false** 行が削除されました。

   CR の例:

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovOperatorConfig
   metadata:
     name: default
     namespace: openshift-sriov-network-operator
   spec:
     configDaemonNodeSelector:
       "node-role.kubernetes.io/$mcp": ""
     disableDrain: true
     enableInjector: true
     enableOperatorWebhook: true
   ```

2. **group-du-sno-ranGen.yaml** ファイル内で、影響を受けるポリシーの **complianceType** を **mustonlyhave** に変更します。

   サンプル YAML

```
# ...
policyDefaults:
  complianceType: "mustonlyhave"
# ...
policies:
  - name: config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: ""
    manifests:
      - path: source-crs/SriovOperatorConfig.yaml
```

3. **ClusterGroupUpdates** CR を作成し、CR の変更を受け取る必要があるクラスターを指定します。

   ClusterGroupUpdates CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-remove
  namespace: default
spec:
  managedPolicies:
    - ztp-group.group-du-sno-config-policy
  enable: false
  clusters:
  - spoke1
  - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:
```

4. 以下のコマンドを実行して **ClusterGroupUpgrade** CR を作成します。

```
$ oc create -f cgu-remove.yaml
```

5. たとえば適切なメンテナンス期間中などに変更を適用する準備が完了したら、次のコマンドを実行して **spec.enable** フィールドの値を **true** に変更します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

検証

1. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get <kind> <changed_cr_name>
```

出力例

```
NAMESPACE   NAME                                        REMEDIATION ACTION
COMPLIANCE STATE   AGE
```

```
default     cgu-ztp-group.group-du-sno-config-policy     enforce                17m
default     ztp-group.group-du-sno-config-policy         inform          NonCompliant
15h
```

ポリシーの **COMPLIANCE STATE** が **Compliant** の場合、CR が更新され、不要なコンテンツが削除されたことを意味します。

2. マネージドクラスターで次のコマンドを実行して、対象クラスターからポリシーが削除されたことを確認します。

```
$ oc get <kind> <changed_cr_name>
```

結果がない場合、CR はマネージドクラスターから削除されます。

9.1.10. GitOps ZTP インストール完了の表示

GitOps Zero Touch Provisioning (ZTP) は、クラスターの GitOps ZTP インストールステータスを確認するプロセスを単純化します。GitOps ZTP ステータスは、クラスターのインストール、クラスター設定、GitOps ZTP 完了の 3 つのフェーズを遷移します。

クラスターインストールフェーズ

クラスターのインストールフェーズは、**ManagedCluster** CR の **ManagedClusterJoined** および **ManagedClusterAvailable** 条件によって示されます。**ManagedCluster** CR にこの条件がない場合や、条件が **False** に設定されている場合、クラスターはインストールフェーズに残ります。インストールに関する追加情報は、**AgentClusterInstall** および **ClusterDeployment** CR から入手できます。詳細は、Troubleshooting GitOps ZTP を参照してください。

クラスター設定フェーズ

クラスター設定フェーズは、クラスターの **ManagedCluster** CR に適用される **ztp-running** ラベルで示されます。

GitOps ZTP 完了

クラスターのインストールと設定は、GitOps ZTP 完了フェーズで実行されます。これは、**ztp-running** ラベルを削除し、**ManagedCluster** CR に **ztp-done** ラベルを追加することで表示されます。**ztp-done** ラベルは、設定が適用され、ベースライン DU 設定が完了したことを示しています。ZTP 完了状態への遷移は、Red Hat Advanced Cluster Management (RHACM) バリデーターのインフォームドポリシーの準拠状態が条件となります。このポリシーは、完了したインストールの既存の基準をキャプチャし、マネージドクラスターの GitOps ZTP プロビジョニングが完了したときにのみ、準拠した状態に移行することを検証するものです。

バリデータ通知ポリシーは、クラスターの設定が完全に適用され、Operator が初期化を完了したことを確認します。ポリシーは以下を検証します。

- ターゲット **MachineConfigPool** には予想されるエントリーが含まれ、更新が完了しました。全ノードが利用可能で、低下することはありません。
- SR-IOV Operator は、**syncStatus: Succeeded** の 1 つ以上の **SriovNetworkNodeState** によって示されているように初期化を完了しています。
- PTP Operator デーモンセットが存在する。

9.2. POLICYGENERATOR リソースを使用した高度なマネージドクラスター設定

PolicyGenerator CR を使用して、マネージドクラスターにカスタム機能をデプロイできます。

重要

GitOps ZTP での PolicyGenerator リソースの使用は、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 製品サポート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場合があります。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。

注記

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

9.2.1. 追加の変更のクラスターへのデプロイ

基本の GitOps Zero Touch Provisioning (ZTP) パイプライン設定以外のクラスター設定を変更する必要がある場合、次の 3 つのオプションを実行できます。

ZTP パイプラインの完了後に追加設定を適用する

GitOps ZTP パイプラインのデプロイが完了すると、デプロイされたクラスターはアプリケーションのワークロードに対応できるようになります。この時点で、Operator を追加インストールし、お客様の要件に応じた設定を適用することができます。追加のコンフィギュレーションがプラットフォームのパフォーマンスや割り当てられた CPU バジェットに悪影響を与えないことを確認する。

GitOps ZTP ライブラリーにコンテンツを追加する

GitOps ZTP パイプラインでデプロイするベースソースのカスタムリソース (CR) は、必要に応じてカスタムコンテンツで拡張できます。

クラスターインストール用の追加マニフェストの作成

インストール時に余分なマニフェストが適用され、インストール作業を効率化することができます。

重要

追加のソース CR を提供したり、既存のソース CR を変更したりすると、OpenShift Container Platform のパフォーマンスまたは CPU プロファイルに大きな影響を与える可能性があります。

関連情報

- GitOps ZTP パイプラインでの追加インストールマニフェストのカスタマイズ

9.2.2. PolicyGenerator CR を使用して、ソース CR の内容をオーバーライドする

PolicyGenTemplate カスタムリソース (CR) を使用すると、**ztp-site-generate** コンテナーの GitOps プラグインで提供されるベースソース CR の上に追加の設定の詳細をオーバーレイできます。**PolicyGenerator** CR は、ベース CR の論理マージまたはパッチと考えることができます。

PolicyGenTemplate CR を使用して、ベース CR の単一フィールドを更新するか、ベース CR の内容全体をオーバーレイします。ベース CR にない値の更新やフィールドの挿入が可能です。

以下の手順例では、**group-du-sno-ranGen.yaml** ファイル内の **PolicyGenTemplate** CR に基づいて、参照設定用に生成された **PerformanceProfile** CR のフィールドを更新する方法について説明します。この手順を元に、PolicyGenTemplate の 他の部分をお客様のご要望に応じて変更してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。

手順

1. 既存のコンテンツのベースラインソース CR を確認します。参照 **PolicyGenTemplate** CR に記載されているソース CR を GitOps Zero Touch Provisioning (ZTP) コンテナーから抽出し、確認すできます。

   a. /**out** フォルダーを作成します。

   ```
   $ mkdir -p ./out
   ```

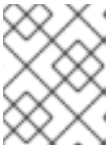
   b. ソース CR を抽出します。

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16.1 extract /home/ztp --tar | tar x -C ./out
   ```

2. **./out/source-crs/PerformanceProfile.yaml** にあるベースライン **PerformanceProfile** CR を確認します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
```

```
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```



注記

ソース CR のフィールドで **$...** を含むものは、**PolicyGenTemplate** CR で提供されない場合、生成された CR から削除されます。

3. **acm-group-du-sno-ranGen.yaml** 参照ファイルの **PerformanceProfile** の **PolicyGenerator** エントリーを更新します。次の例の PolicyGenTemplate CR スタンザは、適切な CPU 仕様を提供し、hugepages 設定を設定し、globallyDisableIrqLoadBalancing を false に設定する新しいフィールドを追加しています。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        # These must be tailored for the specific hardware platform
        cpu:
          isolated: "2-19,22-39"
          reserved: "0-1,20-21"
        hugepages:
          defaultHugepagesSize: 1G
          pages:
          - size: 1G
            count: 10
        globallyDisableIrqLoadBalancing: false
```

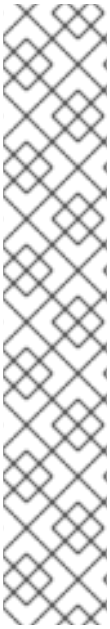
4. Git で **PolicyGenTemplate** 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

出力例

GitOps ZTP アプリケーションは、生成された **PerformanceProfile** CR を含む RHACM ポリシーを生成します。この CR の内容は， **PolicyGenTemplate** の **PerformanceProfile** エントリーから metadata と **spec** の内容をソース CR にマージすることで得られるものである．作成される CR には以下のコンテンツが含まれます。

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: openshift-node-performance-profile
spec:
    additionalKernelArgs:
        - idle=poll
        - rcupdate.rcu_normal_after_boot=0
    cpu:
        isolated: 2-19,22-39
        reserved: 0-1,20-21
    globallyDisableIrqLoadBalancing: false
    hugepages:
        defaultHugepagesSize: 1G
```

```
    pages:
      - count: 10
        size: 1G
machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
net:
    userLevelNetworking: true
nodeSelector:
    node-role.kubernetes.io/master: ""
numa:
    topologyPolicy: restricted
realTimeKernel:
    enabled: true
```



注記

ztp-site-generate コンテナーからデプロイメントした **/source-crs** フォルダーでは、**$** 構文が暗示するテンプレート置換は使用されません。むしろ、policyGen ツールが文字列の $ 接頭辞を認識し、関連する PolicyGenTemplate CR でそのフィールドの値を指定しない場合、そのフィールドは出力 CR から完全に省かれます。

例外として、**/source-crs** YAML ファイル内の **$mcp** 変数は、**PolicyGenTemplate** CR から **mcp の** 指定値で代用されます。例えば、example/policygentemplates/group-du-standard-ranGen.yaml では、mcp の 値は worker となって います。

```
spec:
  bindingRules:
    group-du-standard: ""
  mcp: "worker"
```

policyGen ツールは、**$mcp** のインスタンスを出力 CR の **worker** に置き換えます。

9.2.3. GitOps ZTP パイプラインへのカスタムコンテンツの追加

GitOps ZTP パイプラインに新しいコンテンツを追加するには、次の手順を実行します。

手順

1. **PolicyGenTemplate** カスタムリソース (CR) の **kustomization.yaml** ファイルが含まれるディレクトリーに、**source-crs** という名前のサブディレクトリーを作成します。

2. 次の例に示すように、ユーザー提供の CR を **source-crs** サブディレクトリーに追加します。

```
example
└── acmpolicygenerator
    ├── dev.yaml
    ├── kustomization.yaml
    ├── mec-edge-sno1.yaml
    ├── sno.yaml
    └── source-crs 1
        ├── PaoCatalogSource.yaml
        ├── PaoSubscription.yaml
        ├── custom-crs
        |   ├── apiserver-config.yaml
```

```
|   └── disable-nic-lldp.yaml
└── elasticsearch
    ├── ElasticsearchNS.yaml
    └── ElasticsearchOperatorGroup.yaml
```

1. **source-crs** サブディレクトリーは、**kustomization.yaml** ファイルと同じディレクトリーにある必要があります。

3. 必要な **PolicyGenTemplate** CR を更新して、**source-crs/custom-crs** および **source-crs/elasticsearch** ディレクトリーに追加したコンテンツへの参照を含めます。以下に例を示します。

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-dev
placementBindingDefaults:
  name: group-dev-placement-binding
policyDefaults:
  namespace: ztp-clusters
  placement:
    labelSelector:
      matchExpressions:
        - key: dev
          operator: In
          values:
            - "true"
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: group-dev-group-dev-cluster-log-ns
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogNS.yaml
  - name: group-dev-group-dev-cluster-log-operator-group
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogOperGroup.yaml
  - name: group-dev-group-dev-cluster-log-sub
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/ClusterLogSubscription.yaml
  - name: group-dev-group-dev-lso-ns
    policyAnnotations:
```

```
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/StorageNS.yaml
- name: group-dev-group-dev-lso-operator-group
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/StorageOperGroup.yaml
- name: group-dev-group-dev-lso-sub
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/StorageSubscription.yaml
- name: group-dev-group-dev-pao-cat-source
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "1"
  manifests:
    - path: source-crs/PaoSubscriptionCatalogSource.yaml
      patches:
        - spec:
            image: <container_image_url>
- name: group-dev-group-dev-pao-ns
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/PaoSubscriptionNS.yaml
- name: group-dev-group-dev-pao-sub
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/PaoSubscription.yaml
- name: group-dev-group-dev-elasticsearch-ns
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: elasticsearch/ElasticsearchNS.yaml (1)
- name: group-dev-group-dev-elasticsearch-operator-group
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: elasticsearch/ElasticsearchOperatorGroup.yaml
- name: group-dev-group-dev-apiserver-config
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: custom-crs/apiserver-config.yaml (2)
- name: group-dev-group-dev-disable-nic-lldp
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: custom-crs/disable-nic-lldp.yaml
```

(1) (2) **policies.manifests.path** を設定して、親ディレクトリーからのファイルへの相対パスを追加します。

4. Git で PolicyGenTemplate の変更をコミットし、GitOps ZTP Argo CD ポリシーアプリケーションが監視する Git リポジトリーにプッシュします。

5. ClusterGroupUpgrade CR を更新して、変更された PolicyGenTemplate を含め、cgu-test.yaml として保存します。次の例は、生成された cgu-test.yaml ファイルを示しています。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: custom-source-cr
  namespace: ztp-clusters
spec:
  managedPolicies:
    - group-dev-config-policy
  enable: true
  clusters:
  - cluster1
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

6. 次のコマンドを実行して、更新された ClusterGroupUpgrade CR を適用します。

```
$ oc apply -f cgu-test.yaml
```

検証

- 次のコマンドを実行して、更新が成功したことを確認します。

```
$ oc get cgu -A
```

出力例

```
NAMESPACE     NAME             AGE   STATE       DETAILS
ztp-clusters  custom-source-cr  6s    InProgress  Remediating non-compliant policies
ztp-install   cluster1         19h   Completed   All clusters are compliant with all the
managed policies
```

9.2.4. PolicyGenerator CR のポリシーコンプライアンス評価タイムアウトの設定

ハブクラスターにインストールされた Red Hat Advanced Cluster Management (RHACM) を使用して、管理対象クラスターが適用されたポリシーに準拠しているかどうかを監視および報告します。RHACM は、ポリシーテンプレートを使用して、定義済みのポリシーコントローラーとポリシーを適用します。ポリシーコントローラーは Kubernetes のカスタムリソース定義 (CRD) インスタンスです。

デフォルトのポリシー評価間隔は、PolicyGenTemplate カスタムリソース (CR) でオーバーライドできます。RHACM が適用されたクラスターポリシーを再評価する前に、ConfigurationPolicy CR がポリシー準拠または非準拠の状態を維持できる期間を定義する期間設定を設定します。

GitOps Zero Touch Provisioning (ZTP) ポリシージェネレーターは、事前定義されたポリシー評価間隔で ConfigurationPolicy CR ポリシーを生成します。noncompliant 状態のデフォルト値は 10 秒です。compliant 状態のデフォルト値は 10 分です。評価間隔を無効にするには、値を never に設定します。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. PolicyGenerator CR のすべてのポリシーの評価間隔を設定するには、evaluationInterval フィールドに適切な compliant 値と noncompliant 値を設定します。以下に例を示します。

```
policyDefaults:
  evaluationInterval:
    compliant: 30m
    noncompliant: 45s
```

注記

また、準拠フィールドと 非 準拠 フィールドを never に設定して、特定のコンプライアンス状態に達した後にポリシーの評価を停止することもできます。

2. PolicyGenerator CR の個々のポリシーオブジェクトの評価間隔を設定するに

は、evaluationInterval フィールドを追加し、適切な値を設定します。以下に例を示します。

```
policies:
  - name: "sriov-sub-policy"
    manifests:
      - path: "SriovSubscription.yaml"
        evaluationInterval:
          compliant: never
          noncompliant: 10s
```

3. PolicyGenerator CR ファイルを Git リポジトリーにコミットし、変更をプッシュします。

検証

マネージドスポーククラスターポリシーが予想される間隔で監視されていることを確認します。

1. 管理対象クラスターで cluster-admin 権限を持つユーザーとしてログインします。

2. open-cluster-management-agent-addon namespace で実行されている Pod を取得します。以下のコマンドを実行します。

```
$ oc get pods -n open-cluster-management-agent-addon
```

出力例

```
NAME                                READY   STATUS    RESTARTS       AGE
config-policy-controller-858b894c68-v4xdb   1/1     Running   22 (5d8h ago)   10d
```

3. config-policy-controller Pod のログで、適用されたポリシーが予想される間隔で評価されていることを確認します。

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-
858b894c68-v4xdb
```

出力例

```
2022-05-10T15:10:25.280Z	info	configuration-policy-controller	controllers/configurationpolicy_controller.go:166	Skipping the policy evaluation due to the policy not reaching the evaluation interval	{"policy": "compute-1-config-policy-config"}
2022-05-10T15:10:25.280Z	info	configuration-policy-controller	controllers/configurationpolicy_controller.go:166	Skipping the policy evaluation due to the policy not reaching the evaluation interval	{"policy": "compute-1-common-compute-1-catalog-policy-config"}
```

9.2.5. バリデーターインフォームポリシーを使用した GitOps ZTP クラスターデプロイメントの完了のシグナリング

デプロイされたクラスターの GitOps Zero Touch Provisioning (ZTP) のインストールと設定が完了したときに通知するバリデーター通知ポリシーを作成します。このポリシーは、単一ノード OpenShift クラスター、3 ノードクラスター、および標準クラスターのデプロイメントに使用できます。

手順

1. ソースファイル validatorCRs/informDuValidator.yaml を含むスタンドアロンの PolicyGenTemplate カスタムリソース (CR) を作成します。スタンドアロン PolicyGenerator CR は、各クラスタータイプに 1 つだけ必要です。たとえば、次の CR は、単一ノードの OpenShift クラスターにバリデータ通知ポリシーを適用します。

単一ノードクラスターバリデータ通知ポリシー CR の例 (group-du-sno-validator-ranGen.yaml)

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno-validator-latest
placementBindingDefaults:
  name: group-du-sno-validator-latest-placement-binding
policyDefaults:
  namespace: ztp-group
  placement:
    labelSelector:
      matchExpressions:
        - key: du-profile
          operator: In
          values:
            - latest
        - key: group-du-sno
          operator: Exists
        - key: ztp-done
```

```
            operator: DoesNotExist
    remediationAction: inform
    severity: low
    namespaceSelector:
      exclude:
        - kube-*
      include:
        - '*'
    evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
  - name: group-du-sno-validator-latest-du-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "10000"
    evaluationInterval:
      compliant: 5s
    manifests:
      - path: source-crs/validatorCRs/informDuValidator-MCP-master.yaml
```

2. PolicyGenerator CR ファイルを Git リポジトリーにコミットし、変更をプッシュします。

関連情報

- GitOps ZTP のアップグレード

9.2.6. PolicyGenerator CR を使用した電源状態の設定

低レイテンシーで高パフォーマンスのエッジデプロイメントでは、C ステートと P ステートを無効にするか制限する必要があります。この設定では、CPU は一定の周波数 (通常は最大ターボ周波数) で実行されます。これにより、CPU が常に最大速度で実行され、高いパフォーマンスと低レイテンシーが実現されます。これにより、ワークロードのレイテンシーが最適化されます。ただし、これは最大の電力消費にもつながり、すべてのワークロードに必要ではない可能性があります。

ワークロードはクリティカルまたは非クリティカルとして分類できます。クリティカルなワークロードでは、高パフォーマンスと低レイテンシーのために C ステートと P ステートの設定を無効にする必要があります。クリティカルでないワークロードでは、C ステートと P ステートの設定を使用して、いくらかのレイテンシーとパフォーマンスを犠牲にします。GitOps Zero Touch Provisioning (ZTP) を使用して、次の 3 つの電源状態を設定できます。

- 高性能モードは、最大の消費電力で超低遅延を提供します。

- パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。
- 省電力は、消費電力の削減と遅延の増加のバランスをとります。

デフォルトの設定は、低遅延のパフォーマンスモードです。

PolicyGenTemplate カスタムリソース (CR) を使用すると、ztp-site-generate コンテナーの GitOps プラグインで提供されるベースソース CR に追加の設定の詳細をオーバーレイできます。

group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して、電源状態を設定します。

次の共通の前提条件は、3 つの電源状態すべての設定に適用されます。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。
- GitOps ZTP サイト設定リポジトリーの準備で説明されている手順に従っていること。

関連情報

- ワークロードヒントを使用したノードの電力消費とリアルタイム処理の設定

9.2.6.1. PolicyGenerator CR を使用したパフォーマンスモードの設定

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新してパフォーマンスモードを設定します。

パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

1. out/argocd/example/acmpolicygenerator// にある acm-group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenerator エントリーを更新して、パフォーマンスモードを設定します。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        workloadHints:
          realTime: true
          highPowerConsumption: false
          perPodPowerManagement: false
```

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

9.2.6.2. PolicyGenerator CR を使用した高パフォーマンスモードの設定

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して高パフォーマンスモードを設定します。

高パフォーマンスモードは、最大の消費電力で超低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

1. 高パフォーマンスモードを設定するには、次のように out/argocd/example/acmpolicygenerator/ にある acm-group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenerator エントリーを更新します。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        workloadHints:
          realTime: true
          highPowerConsumption: true
          perPodPowerManagement: false
```

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

9.2.6.3. PolicyGenerator CR を使用した省電力モードの設定

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して、省電力モードを設定します。

省電力モードは、消費電力の削減と遅延の増加のバランスをとります。

前提条件

- BIOS で C ステートと OS 制御の P ステートを有効にしました。

手順

1. 省電力モードを設定するには、次のように out/argocd/example/acmpolicygenerator/ にある acm-group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenerator エントリーを更新します。追加のカーネル引数オブジェクトを使用して、省電力モード用に CPU ガバナーを設定することを推奨します。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        # ...
        workloadHints:
          realTime: true
          highPowerConsumption: false
          perPodPowerManagement: true
        # ...
        additionalKernelArgs:
          - # ...
          - "cpufreq.default_governor=schedutil" (1)
```

(1)

schedutil ガバナーが推奨されますが、ondemand や powersave などの他のガバナーを使用することもできます。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

検証

1. 次のコマンドを使用して、識別されたノードのリストから、デプロイされたクラスター内のワーカーノードを選択します。

```
$ oc get nodes
```

2. 次のコマンドを使用して、ノードにログインします。

```
$ oc debug node/<node-name>
```

<node-name> を、電源状態を確認するノードの名前に置き換えます。

3. /host をデバッグシェル内の root ディレクトリーとして設定します。デバッグ Pod は、Pod 内の /host にホストの root ファイルシステムをマウントします。次の例に示すように、ルートディレクトリーを /host に変更すると、ホストの実行可能パスに含まれるバイナリーを実行できます。

```
# chroot /host
```

4. 次のコマンドを実行して、適用された電源状態を確認します。

```
# cat /proc/cmdline
```

予想される出力

- 省電力モードの intel_pstate=passive。

関連情報

-

高優先度のワークロードと低優先度のワークロードを同じ場所で実行するノードの省電力設定

- 低遅延と高パフォーマンスのためのホストファームウェアの設定
- GitOps ZTP サイト設定リポジトリーの準備

9.2.6.4. 省電力の最大化

最大の CPU 周波数を制限して、最大の電力節約を実現することを推奨します。最大 CPU 周波数を制限せずに重要でないワークロード CPU で C ステートを有効にすると、重要な CPU の周波数が高くなるため、消費電力の節約の多くが無効になります。

sysfs プラグインフィールドを更新し、リファレンス設定の TunedPerformancePatch CR で max_perf_pct に適切な値を設定することで、電力の節約を最大化します。group-du-sno-ranGen.yaml に基づくこの例では、最大 CPU 周波数を制限するために従う手順について説明します。

前提条件

- PolicyGenTemplate CR を使用した省電力モードの設定の説明に従って、省電力モードを設定しました。

手順

1. out/argocd/example/acmpolicygenerator/ の acm-group-du-sno-ranGen.yaml 参照ファイルで、TunedPerformancePatch の PolicyGenerator エントリーを更新します。電力を最大限に節約するには、次の例に示すように max_perf_pct を追加します。

```
- path: source-crs/TunedPerformancePatch.yaml
  patches:
    - spec:
      profile:
        - name: performance-patch
          data: |
            # ...
            [sysfs]
            /sys/devices/system/cpu/intel_pstate/max_perf_pct=<x> 1
```

1. max_perf_pct は、cpufreq ドライバーが設定できる最大周波数を、サポートされている最大 CPU 周波数のパーセンテージとして制御します。この値はすべての CPU に適用

されます。サポートされている最大周波数は /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq で確認できます。開始点として、All Cores Turbo 周波数ですべての CPU を制限する割合を使用できます。All Cores Turbo 周波数は、すべてのコアがすべて使用されているときに全コアが実行される周波数です。

注記

省電力を最大化するには、より低い値を設定します。max_perf_pct の値を低く設定すると、最大 CPU 周波数が制限されるため、消費電力が削減されますが、パフォーマンスに影響を与える可能性もあります。さまざまな値を試し、システムのパフォーマンスと消費電力を監視して、ユースケースに最適な設定を見つけてください。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

9.2.7. PolicyGenerator CR を使用した LVM ストレージの設定

GitOps Zero Touch Provisioning (ZTP) を使用して、デプロイするマネージドクラスターの論理ボリュームマネージャー (LVM) ストレージを設定できます。

注記

HTTP トランスポートで PTP イベントまたはベアメタルハードウェアイベントを使用する場合、LVM ストレージを使用してイベントサブスクリプションを永続化します。

分散ユニットでローカルボリュームを使用する永続ストレージには、Local Storage Operator を使用します。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。

手順

1. 新しいマネージドクラスター用に LVM ストレージを設定するには、次の YAML を acm-common-ranGen.yaml ファイルの policies.manifests に追加します。

```
- name: subscription-policies
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/StorageLVMOSubscriptionNS.yaml
    - path: source-crs/StorageLVMOSubscriptionOperGroup.yaml
    - path: source-crs/StorageLVMOSubscription.yaml
      spec:
        name: lvms-operator
        channel: stable-4.16
```

注記

Storage LVMO サブスクリプションは非推奨になりました。OpenShift Container Platform の将来のリリースでは、ストレージ LVMO サブスクリプションは利用できなくなります。代わりに、Storage LVMS サブスクリプションを使用する必要があります。

OpenShift Container Platform 4.15 では、LVMO サブスクリプションの代わりに Storage LVMS サブスクリプションを使用できます。LVMS サブスクリプションでは、common-ranGen.yaml ファイルを手動で上書きする必要はありません。次の YAML を acm-common-ranGen.yaml ファイルの policies.manifests に追加して、Storage LVMS サブスクリプションを使用します。

```
- path: source-crs/StorageLVMSubscriptionNS.yaml
- path: source-crs/StorageLVMSubscriptionOperGroup.yaml
- path: source-crs/StorageLVMSubscription.yaml
```

2. 特定のグループまたは個々のサイト設定ファイルの policies.manifests に LVMCluster CR を追加します。たとえば、group-du-sno-ranGen.yaml ファイルに次を追加します。

```
- fileName: StorageLVMCluster.yaml
  policyName: "lvms-config"
    metadata:
      name: "lvms-storage-cluster-config"
        spec:
```

```
storage:
  deviceClasses:
  - name: vg1
    thinPoolConfig:
      name: thin-pool-1
      sizePercent: 90
      overprovisionRatio: 10
```

この設定例では、OpenShift Container Platform がインストールされているディスクを除く、使用可能なすべてのデバイスを含むボリュームグループ (vg1) を作成します。シンプール論理ボリュームも作成されます。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

4. Git で PolicyGenTemplate の変更をコミットし、その変更をサイト設定リポジトリーにプッシュして、GitOps ZTP を使用して LVM ストレージを新しいサイトにデプロイします。

9.2.8. PolicyGenerator CR を使用した PTP イベントの設定

GitOps ZTP パイプラインを使用して、HTTP または AMQP トランスポートを使用する PTP イベントを設定できます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

9.2.8.1. HTTP トランスポートを使用する PTP イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、HTTP トランスポートを使用する PTP イベントを設定できます。

前提条件

- OpenShift CLI (oc) がインストールされている。

- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. 要件に応じて、次の PolicyGenerator の変更を acm-group-du-3node-ranGen.yaml ファイル、acm-group-du-sno-ranGen.yaml ファイル、または acm-group-du-standard-ranGen.yaml ファイルに適用します。

   a. policies.manifests に、トランスポートホストを設定する PtpOperatorConfig CR ファイルを追加します。

```
- path: source-crs/PtpOperatorConfigForEvent.yaml
  patches:
  - metadata:
      name: default
      namespace: openshift-ptp
      annotations:
        ran.openshift.io/ztp-deploy-wave: "10"
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/$mcp: ""
      ptpEventConfig:
        enableEventPublisher: true
        transportHost: "http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043"
```

注記

OpenShift Container Platform 4.13 以降では、PTP イベントに HTTP トランスポートを使用するときに、PtpOperatorConfig リソースの transportHost フィールドを設定する必要はありません。

   b. PTP クロックの種類とインターフェイスに linuxptp と phc2sys を設定します。たとえば、以下の YAML を policies.manifests に追加します。

```
- path: source-crs/PtpConfigSlave.yaml (1)
  patches:
  - metadata:
      name: "du-ptp-slave"
    spec:
      recommend:
```

```
- match:
  - nodeLabel: node-role.kubernetes.io/master
  priority: 4
  profile: slave
profile:
- name: "slave"
  # This interface must match the hardware in this group
  interface: "ens5f0" (2)
  ptp4lOpts: "-2 -s --summary_interval -4" (3)
  phc2sysOpts: "-a -r -n 24" (4)
  ptpSchedulingPolicy: SCHED_FIFO
  ptpSchedulingPriority: 10
  ptpSettings:
    logReduce: "true"
  ptp4lConf: |
    [global]
    #
    # Default Data Set
    #
    twoStepFlag 1
    slaveOnly 1
    priority1 128
    priority2 128
    domainNumber 24
    #utc_offset 37
    clockClass 255
    clockAccuracy 0xFE
    offsetScaledLogVariance 0xFFFF
    free_running 0
    freq_est_interval 1
    dscp_event 0
    dscp_general 0
    dataset_comparison G.8275.x
    G.8275.defaultDS.localPriority 128
    #
    # Port Data Set
    #
    logAnnounceInterval -3
    logSyncInterval -4
    logMinDelayReqInterval -4
    logMinPdelayReqInterval -4
    announceReceiptTimeout 3
    syncReceiptTimeout 0
    delayAsymmetry 0
    fault_reset_interval -4
    neighborPropDelayThresh 20000000
    masterOnly 0
    G.8275.portDS.localPriority 128
    #
    # Run time options
    #
    assume_two_step 0
    logging_level 6
    path_trace_enabled 0
    follow_up_info 0
    hybrid_e2e 0
```

```
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
```

```
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
    ptpClockThreshold: 5
      holdOverTimeout: 30 # seconds
      maxOffsetThreshold: 100  # nano seconds
      minOffsetThreshold: -100
```

1

必要に応じて、PtpConfigMaster.yaml、PtpConfigSlave.yaml、または PtpConfigSlaveCvl.yaml のいずれか 1 つを指定できます。PtpConfigSlaveCvl.yaml は、Intel E810 Columbiaville NIC の linuxptp サービスを設定します。group-du-sno-ranGen.yaml および group-du-3node-ranGen.yaml に基づいて設定する場合は、PtpConfigSlave.yaml を使用します。

2

デバイス固有のインターフェイス名。

3

PTP 高速イベントを有効にするには、.spec.sourceFiles.spec.profile の ptp4lOpts に --summary_interval -4 値を追加する必要があります。

4

phc2sysOpts の値が必要です。-m はメッセージを stdout に出力します。linuxptp-daemon DaemonSet はログを解析し、Prometheus メトリックを生成します。

5

オプション: ptpClockThreshold スタンザが存在しない場合は、ptpClockThreshold フィールドにデフォルト値が使用されます。スタンザは、デフォルトの ptpClockThreshold 値を示します。ptpClockThreshold 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。holdOverTimeout は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が FREERUN に変わるまでの時間値 (秒単位) です。maxOffsetThreshold および minOffsetThreshold 設定は、CLOCK_REALTIME (phc2sys) またはマスターオフセット (ptp4l) の値と比較するナノ秒単位のオフセット値を設定します。ptp4l または phc2sys のオフセット値がこの範囲外の場合、PTP クロックの状態が FREERUN に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が LOCKED に設定されます。

2. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

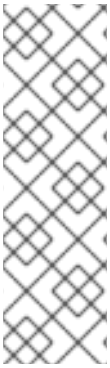
3. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- PolicyGenerator CR を使用して、ソース CR の内容をオーバーライドする

9.2.8.2. AMQP トランスポートを使用する PTP イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイするマネージドクラスター上で、AMQP トランスポートを使用する PTP イベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. 次の YAML を acm-common-ranGen.yaml ファイルの policies.manifests に追加して、AMQP Operator を設定します。

```
#AMQ Interconnect Operator for fast events
- path: source-crs/AmqSubscriptionNS.yaml
- path: source-crs/AmqSubscriptionOperGroup.yaml
- path: source-crs/AmqSubscription.yaml
```

2. 要件に応じて、次の PolicyGenerator の変更を acm-group-du-3node-ranGen.yaml ファイル、acm-group-du-sno-ranGen.yaml ファイル、または acm-group-du-standard-ranGen.yaml ファイルに適用します。

   a. policies.manifests に、AMQ トランスポートホストを設定する PtpOperatorConfig CR ファイルを config-policy に追加します。

```
- path: source-crs/PtpOperatorConfigForEvent.yaml
  patches:
  - metadata:
      name: default
      namespace: openshift-ptp
      annotations:
        ran.openshift.io/ztp-deploy-wave: "10"
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/$mcp: ""
      ptpEventConfig:
        enableEventPublisher: true
        transportHost: "amqp://amq-router.amq-router.svc.cluster.local"
```

   b. PTP クロックの種類とインターフェイスに linuxptp と phc2sys を設定します。たとえば、以下の YAML を policies.manifests に追加します。

```
- path: source-crs/PtpConfigSlave.yaml (1)
  patches:
  - metadata:
      name: "du-ptp-slave"
    spec:
      recommend:
      - match:
        - nodeLabel: node-role.kubernetes.io/master
        priority: 4
        profile: slave
      profile:
      - name: "slave"
        # This interface must match the hardware in this group
        interface: "ens5f0" (2)
        ptp4lOpts: "-2 -s --summary_interval -4" (3)
        phc2sysOpts: "-a -r -n 24" (4)
        ptpSchedulingPolicy: SCHED_FIFO
        ptpSchedulingPriority: 10
        ptpSettings:
          logReduce: "true"
        ptp4lConf: |
          [global]
          #
          # Default Data Set
          #
```

```
twoStepFlag 1
slaveOnly 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 255
clockAccuracy 0xFE
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
```

```
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  ptpClockThreshold: 5
    holdOverTimeout: 30 # seconds
    maxOffsetThreshold: 100  # nano seconds
    minOffsetThreshold: -100
```

1

必要に応じて、PtpConfigMaster.yaml、PtpConfigSlave.yaml、または PtpConfigSlaveCvl.yaml のいずれか 1 つを指定できます。PtpConfigSlaveCvl.yaml は、Intel E810 Columbiaville NIC の linuxptp サービスを設定します。group-du-sno-ranGen.yaml および group-du-3node-ranGen.yaml に基づいて設定する場合は、PtpConfigSlave.yaml を使用します。

2

デバイス固有のインターフェイス名。

3

PTP 高速イベントを有効にするには、.spec.sourceFiles.spec.profile の ptp4lOpts に --summary_interval -4 値を追加する必要があります。

4

phc2sysOpts の値が必要です。-m はメッセージを stdout に出力します。linuxptp-daemon DaemonSet はログを解析し、Prometheus メトリックを生成します。

5

オプション: ptpClockThreshold スタンザが存在しない場合は、ptpClockThreshold フィールドにデフォルト値が使用されます。スタンザは、デフォルトの ptpClockThreshold 値を示します。ptpClockThreshold 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。holdOverTimeout は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が FREERUN に変わるまでの時間値 (秒単位) です。maxOffsetThreshold および minOffsetThreshold 設定は、CLOCK_REALTIME (phc2sys) またはマスターオフセット (ptp4l) の値と比較するナノ秒単位のオフセット値を設定します。ptp4l または phc2sys のオフセット値がこの範囲外の場合、PTP クロックの状態が FREERUN に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が LOCKED に設定されます。

3. 以下の PolicyGenerator の変更を、特定のサイトの YAML ファイル（例： acm-example-sno-site.yaml ）に適用します。

   a. policies.manifests に、AMQ ルーターを設定する Interconnect CR ファイルを config-policy に追加します。

   ```
   - path: source-crs/AmqInstance.yaml
   ```

4. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

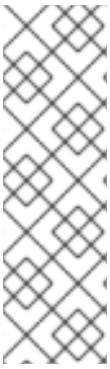
5. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- AMQ メッセージングバスのインストール
- OpenShift イメージレジストリーの概要

9.2.9. PolicyGenerator CR を使用したベアメタルイベントの設定

GitOps ZTP パイプラインを使用して、HTTP または AMQP トランスポートを使用するベアメタルイベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

9.2.9.1. HTTP トランスポートを使用するベアメタルイベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、HTTP トランスポートを使用するベアメタルイベントを設定できます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. 次の YAML を acm-common-ranGen.yaml ファイルの policies.manifests に追加して、Bare Metal Event Relay Operator を設定します。

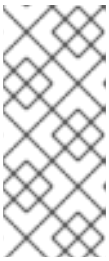
```
# Bare Metal Event Relay Operator
```

```
- path: source-crs/BareMetalEventRelaySubscriptionNS.yaml
- path: source-crs/BareMetalEventRelaySubscriptionOperGroup.yaml
- path: source-crs/BareMetalEventRelaySubscription.yaml
```

2. 特定のグループ設定ファイルの policies.manifests に HardwareEvent CR を追加します（例： acm-group-du-sno-ranGen.yaml ファイル）。

```
- path: source-crs/HardwareEvent.yaml (1)
  patches:
    - spec:
        logLevel: debug
        nodeSelector: {}
        transportHost: http://hw-event-publisher-service.openshift-bare-metal-
events.svc.cluster.local:9043
```

(1) 各ベースボード管理コントローラー (BMC) では、1 つの HardwareEvent CR のみ必要です。



注記

OpenShift Container Platform 4.13 以降では、ベアメタルイベントで HTTP トランスポートを使用する場合、HardwareEvent カスタムリソース (CR) の TransportHost フィールドを設定する必要はありません。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

4. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用してベアメタルイベントを新しいサイトにデプロイします。

5. 次のコマンドを実行して Redfish シークレットを作成します。

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

関連情報

- CLI を使用した Bare Metal Event リレーのインストール
- ベアメタルイベントおよびシークレット CR の作成

9.2.9.2. AMQP トランスポートを使用するベアメタルイベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、AMQP トランスポートを使用するベアメタルイベントを設定できます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. AMQ Interconnect Operator と Bare Metal Event Relay Operator を設定するには、次の YAML を acm-common-ranGen.yaml ファイルの policies.manifests に追加します。

```
# AMQ Interconnect Operator for fast events
- path: source-crs/AmqSubscriptionNS.yaml
- path: source-crs/AmqSubscriptionOperGroup.yaml
- path: source-crs/AmqSubscription.yaml
# Bare Metal Event Relay Operator
- path: source-crs/BareMetalEventRelaySubscriptionNS.yaml
- path: source-crs/BareMetalEventRelaySubscriptionOperGroup.yaml
- path: source-crs/BareMetalEventRelaySubscription.yaml
```

2. Interconnect CR をサイト設定ファイルの policies.manifests に追加します（例： acm-example-sno-site.yaml ファイル）。

```
- path: source-crs/AmqInstance.yaml
```

3. 特定のグループ設定ファイルの policies.manifests に HardwareEvent CR を追加します（例： acm-group-du-sno-ranGen.yaml ファイル）。

```
- path: HardwareEvent.yaml
  patches:
    nodeSelector: {}
    transportHost: "amqp://<amq_interconnect_name>.
<amq_interconnect_namespace>.svc.cluster.local" 1
    logLevel: "info"
```

1. transportHost URL は、既存の AMQ Interconnect CR name と namespace で設定されます。たとえば、transportHost: "amqp://amq-router.amq-router.svc.cluster.local" では、AMQ Interconnect の name と namespace の両方が amq-router に設定されます。

注記

各ベースボード管理コントローラー (BMC) には、単一の HardwareEvent リソースのみが必要です。

4. Git で PolicyGenTemplate の変更をコミットし、その変更をサイト設定リポジトリーにプッシュして、GitOps ZTP を使用してベアメタルイベント監視を新しいサイトにデプロイします。

5. 次のコマンドを実行して Redfish シークレットを作成します。

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

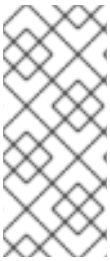
9.2.10. イメージをローカルにキャッシュするための Image Registry Operator の設定

OpenShift Container Platform は、ローカルレジストリーを使用してイメージのキャッシュを管理

します。エッジコンピューティングのユースケースでは、クラスターは集中型のイメージレジストリーと通信するときに帯域幅の制限を受けることが多く、イメージのダウンロード時間が長くなる可能性があります。

初期デプロイメント中はダウンロードに時間がかかることは避けられません。時間の経過とともに、予期しないシャットダウンが発生した場合に CRI-O が /var/lib/containers/storage ディレクトリーを消去するリスクがあります。イメージのダウンロード時間が長い場合の対処方法として、GitOps Zero Touch Provisioning (ZTP) を使用してリモートマネージドクラスター上にローカルイメージレジストリーを作成できます。これは、クラスターがネットワークの遠端にデプロイメントされるエッジコンピューティングシナリオで役立ちます。

GitOps ZTP を使用してローカルイメージレジストリーをセットアップする前に、リモートマネージドクラスターのインストールに使用する SiteConfig CR でディスクパーティショニングを設定する必要があります。インストール後、PolicyGenTemplate CR を使用してローカルイメージレジストリーを設定します。次に、GitOps ZTP パイプラインは永続ボリューム (PV) と永続ボリューム要求 (PVC) CR を作成し、imageregistry 設定にパッチを適用します。



注記

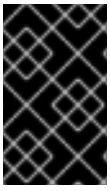
ローカルイメージレジストリーは、ユーザーアプリケーションイメージにのみ使用でき、OpenShift Container Platform または Operator Lifecycle Manager Operator イメージには使用できません。

関連情報

- OpenShift Container Platform レジストリーの概要

9.2.10.1. SiteConfig を使用したディスクパーティショニングの設定

SiteConfig CR と GitOps Zero Touch Provisioning (ZTP) を使用して、マネージドクラスターのディスクパーティションを設定します。SiteConfig CR のディスクパーティションの詳細は、基になるディスクと一致する必要があります。



重要

この手順はインストール時に完了する必要があります。

前提条件

- Butane をインストールします。

手順

1. storage.bu ファイルを作成します。

```
variant: fcos
version: 1.3.0
storage:
 disks:
 - device: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0 (1)
   wipe_table: false
   partitions:
   - label: var-lib-containers
     start_mib: <start_of_partition> (2)
     size_mib: <partition_size> (3)
 filesystems:
   - path: /var/lib/containers
     device: /dev/disk/by-partlabel/var-lib-containers
     format: xfs
     wipe_filesystem: true
     with_mount_unit: true
     mount_options:
       - defaults
       - prjquota
```

(1) ルートディスクを指定します。

(2) パーティションの開始点を MiB 単位で指定します。値が小さすぎると、インストールに失敗します。

(3) パーティションのサイズを指定します。値が小さすぎると、デプロイメントは失敗します。

2. 次のコマンドを実行して、storage.bu を Ignition ファイルに変換します。

```
$ butane storage.bu
```

出力例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# # Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-partlabel/var-lib-containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. JSON Pretty Print などのツールを使用して、出力を JSON 形式に変換します。

4. 出力を SiteConfig CR の .spec.clusters.nodes.ignitionConfigOverride フィールドにコピーします。

例

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: |
          {
            "ignition": {
              "version": "3.2.0"
            },
            "storage": {
              "disks": [
                {
                  "device": "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0",
                  "partitions": [
                    {
                      "label": "var-lib-containers",
                      "sizeMiB": 0,
                      "startMiB": 250000
                    }
                  ],
                  "wipeTable": false
                }
```

```
            ],
            "filesystems": [
              {
                "device": "/dev/disk/by-partlabel/var-lib-containers",
                "format": "xfs",
                "mountOptions": [
                  "defaults",
                  "prjquota"
                ],
                "path": "/var/lib/containers",
                "wipeFilesystem": true
              }
            ]
          },
          "systemd": {
            "units": [
              {
                "contents": "# # Generated by Butane\n[Unit]\nRequires=systemd-
fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-
fsck@dev-disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
                "enabled": true,
                "name": "var-lib-containers.mount"
              }
            ]
          }
        }
[...]
```

注記

.spec.clusters.nodes.ignitionConfigOverride フィールドが存在しない場合は、作成します。

検証

1. インストール時またはインストール後に、次のコマンドを実行して、BareMetalHost オブジェクトがアノテーションを表示することを確認します。

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-
install.openshift.io/ignition-config-overrides"]
```

出力例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":[{\"device\":\"/dev/disk/by-partlabel/var-lib-containers\",\"format\":\"xfs\",\"mountOptions\":[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"systemd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-fsck@dev-disk-by\\\\x2dpartlabel-var\\\\x2dlib\\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-by\\\\x2dpartlabel-var\\\\x2dlib\\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/dev/disk/by-partlabel/var-lib-containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. インストール後に、単一ノードの OpenShift ディスクのステータスを確認します。

a. 次のコマンドを実行して、シングルノード OpenShift ノードのデバッグセッションに入ります。この手順は、<node_name>-debug というデバッグ Pod をインスタンス化します。

```
$ oc debug node/my-sno-node
```

b. 次のコマンドを実行して、デバッグシェル内のルートディレクトリーとして /host を設定します。デバッグ Pod は、Pod 内の /host にホストの root ファイルシステムをマウントします。root ディレクトリーを /host に変更すると、ホストの実行パスに含まれるバイナリーを実行できます。

```
# chroot /host
```

c. 以下のコマンドを実行して、利用可能なすべてのブロックデバイスに関する情報を一覧表示します。

```
# lsblk
```

出力例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                      /sysroot/ostree/deploy/rhcos/var
│                      /usr
│                      /etc
│                      /
│                      /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

d. 次のコマンドを実行して、ファイルシステムのディスク領域の使用状況に関する情報を表示します。

```
# df -h
```

出力例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

9.2.10.2. PolicyGenerator CR を使用したイメージレジストリーの設定

PolicyGenTemplate (PGT) CR を使用して、イメージレジストリーの設定に必要な CR を適用し、imageregistry 設定にパッチを適用します。

前提条件

- 管理対象クラスターでディスクパーティションを設定しました。

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- GitOps Zero Touch Provisioning (ZTP) で使用するカスタムサイト設定データを管理する Git リポジトリーを作成している。

手順

1. 適切な PolicyGenTemplate CR で、ストレージクラス、永続ボリューム要求、永続ボリューム、およびイメージレジストリー設定を設定します。たとえば、個々のサイトを設定するには、次の YAML をファイル example-sno-site.yaml に追加します。

```
sourceFiles:
  # storage class
  - fileName: StorageClass.yaml
    policyName: "sc-for-image-registry"
    metadata:
      name: image-registry-sc
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100" (1)
  # persistent volume claim
  - fileName: StoragePVC.yaml
    policyName: "pvc-for-image-registry"
    metadata:
      name: image-registry-pvc
      namespace: openshift-image-registry
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
      storageClassName: image-registry-sc
      volumeMode: Filesystem
  # persistent volume
  - fileName: ImageRegistryPV.yaml (2)
    policyName: "pv-for-image-registry"
    metadata:
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
  - fileName: ImageRegistryConfig.yaml
    policyName: "config-for-image-registry"
    complianceType: musthave
    metadata:
      annotations:
```

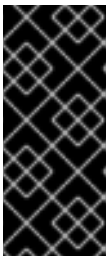
```
    ran.openshift.io/ztp-deploy-wave: "100"
spec:
  storage:
    pvc:
      claim: "image-registry-pvc"
```

1

サイト、共通、またはグループレベルでイメージレジストリーを設定するかどうかに応じて、ztp-deploy-wave に適切な値を設定します。ztp-deploy-wave: "100" は、参照されるソースファイルをグループ化できるため、開発またはテストに適しています。

2

ImageRegistryPV.yaml で、spec.local.path フィールドが /var/imageregistry に設定され、SiteConfig CR の mount_point フィールドに設定された値と一致することを確認します。



重要

- fileName: ImageRegistryConfig.yaml 設定には、complianceType: mustonlyhave を設定しないでください。これにより、レジストリー Pod のデプロイが失敗する可能性があります。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視される Git リポジトリーにプッシュします。

検証

次の手順を使用して、マネージドクラスターのローカルイメージレジストリーに関するエラーをトラブルシューティングします。

- マネージドクラスターにログインしているときに、レジストリーへのログインが成功したことを確認します。以下のコマンドを実行します。

  a. マネージドクラスター名をエクスポートします。

  ```
  $ cluster=<managed_cluster_name>
  ```

  b. マネージドクラスター kubeconfig の詳細を取得します。

```
$ oc get secret -n $cluster $cluster-admin-password -o
jsonpath='{.data.password}' | base64 -d > kubeadmin-password-$cluster
```

c. クラスター kubeconfig をダウンロードしてエクスポートします。

```
$ oc get secret -n $cluster $cluster-admin-kubeconfig -o
jsonpath='{.data.kubeconfig}' | base64 -d > kubeconfig-$cluster && export
KUBECONFIG=./kubeconfig-$cluster
```

d. マネージドクラスターからイメージレジストリーへのアクセスを確認します。レジストリーへのアクセスを参照してください。

- imageregistry.operator.openshift.io グループインスタンスの Config CRD がエラーを報告していないことを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get image.config.openshift.io cluster -o yaml
```

出力例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2021-10-08T19:02:39Z"
  generation: 5
  name: cluster
  resourceVersion: "688678648"
  uid: 0406521b-39c0-4cda-ba75-873697da75a4
spec:
  additionalTrustedCA:
    name: acm-ice
```

- 管理対象クラスターの PersistentVolumeClaim にデータが入力されていることを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get pv image-registry-sc
```

- registry* Pod が実行中であり、openshift-image-registry namespace にあることを確認します。

```
$ oc get pods -n openshift-image-registry | grep registry*
```

出力例

```
cluster-image-registry-operator-68f5c9c589-42cfg   1/1     Running     0          8d
image-registry-5f8987879-6nx6h                     1/1     Running     0          8d
```

- マネージドクラスターのディスクパーティションが正しいことを確認します。

a. マネージドクラスターへのデバッグシェルを開きます。

```
$ oc debug node/sno-1.example.com
```

b. lsblk を実行して、ホストディスクパーティションを確認します。

```
sh-4.4# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
  |-sda1   8:1    0     1M  0 part
  |-sda2   8:2    0   127M  0 part
  |-sda3   8:3    0   384M  0 part /boot
  |-sda4   8:4    0 336.3G  0 part /sysroot
  `-sda5   8:5    0 100.1G  0 part /var/imageregistry 1
sdb      8:16   0 446.6G  0 disk
sr0     11:0    1   104M  0 rom
```

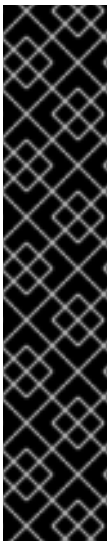
1 /var/imageregistry は、ディスクが正しくパーティショニングされていることを示します。

関連情報

- レジストリーへのアクセス

9.3. POLICYGENERATOR リソースと TALM を使用した非接続環境でのマネージドクラスターの更新

Topology Aware Lifecycle Manager (TALM)を使用して、GitOps Zero Touch Provisioning (ZTP) および Topology Aware Lifecycle Manager (TALM)を使用してデプロイしたマネージドクラスターのソフトウェアライフサイクルを管理できます。TALM は Red Hat Advanced Cluster Management (RHACM) PolicyGenerator ポリシーを使用して、ターゲットクラスターに適用される変更を管理および制御します。



重要

GitOps ZTP での PolicyGenerator リソースの使用は、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 製品サポート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場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

関連情報

- Topology Aware Lifecycle Manager の詳細は、About the Topology Aware Lifecycle Manager を参照してください。

9.3.1. 非接続環境の設定

TALM は、プラットフォームと Operator の更新の両方を実行できます。

TALM を使用して非接続クラスターを更新する前に、ミラーレジストリーで更新するプラットフォームイメージおよび Operator イメージの両方をミラーリングする必要があります。イメージをミラーリングするには以下の手順を実行します。

- プラットフォームの更新では、以下の手順を実行する必要があります。

1. 必要な OpenShift Container Platform イメージリポジトリーをミラーリングしま

す。追加リソースにリンクされている OpenShift Container Platform イメージリポジトリーのミラーリング手順に従って、目的のプラットフォームイメージがミラーリングされていることを確認してください。imageContentSources.yaml ファイルの imageContentSources セクションの内容を保存します。

出力例

```
imageContentSources:
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-release
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. ミラーリングされた目的のプラットフォーム イメージのイメージ シグネチャーを保存します。プラットフォームの更新のために、イメージ署名を PolicyGenerator CR に追加する必要があります。イメージ署名を取得するには、次の手順を実行します。

   a. 以下のコマンドを実行して、目的の OpenShift Container Platform タグを指定します。

   ```
   $ OCP_RELEASE_NUMBER=<release_version>
   ```

   b. 次のコマンドを実行して、クラスターのアーキテクチャーを指定します。

   ```
   $ ARCHITECTURE=<cluster_architecture> 1
   ```

   1 x86_64、aarch64、s390x、または ppc64le など、クラスターのアーキテクチャーを指定します。

   c. 次のコマンドを実行して、Quay からリリースイメージダイジェストを取得します。

```
$ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From: .*@//p')"
```

d. 次のコマンドを実行して、ダイジェストアルゴリズムを設定します。

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

e. 次のコマンドを実行して、ダイジェスト署名を設定します。

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. 次のコマンドを実行して、mirror.openshift.com Web サイトからイメージ署名を取得します。

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

g. 以下のコマンドを実行して、イメージ署名を checksum-<OCP_RELEASE_NUMBER>.yaml ファイルに保存します。

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 更新グラフを準備します。更新グラフを準備するオプションは 2 つあります。

a. OpenShift Update Service を使用します。

ハブクラスターでグラフを設定する方法の詳細については、 OpenShift Update Service の Operator のデプロイ および グラフデータ init コンテナーのビルド を参照してください。

b. アップストリームグラフのローカルコピーを作成します。マネージドクラスターにアクセスできる非接続環境の http または https サーバーで更新グラフをホストします。更新グラフをダウンロードするには、以下のコマンドを使用します。

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.16 -o ~/upgrade-graph_stable-4.16
```

- Operator の更新については、以下のタスクを実行する必要があります。
  - Operator カタログをミラーリングします。切断されたクラスターで使用する Operator カタログのミラーリングセクションの手順に従って、目的の Operator イメージがミラーリングされていることを確認します。

関連情報

- GitOps Zero Touch Provisioning (ZTP) の更新方法について、詳しくは GitOps ZTP のアップグレード を参照してください。
- OpenShift Container Platform イメージリポジトリーをミラーリングする方法の詳細は、OpenShift Container Platform イメージリポジトリーのミラーリング を参照してください。
- 非接続クラスターの Operator カタログをミラーリングする方法の詳細は、非接続クラスターで使用する Operator カタログのミラーリング を参照してください。
- 非接続環境を準備して目的のイメージリポジトリーをミラーリングする方法の詳細は、非接続環境の準備 を参照してください。
- 更新チャネルとリリースの詳細は、更新チャネルとリリースについて を参照してください。

9.3.2. PolicyGenerator CR を使用したプラットフォーム更新の実行

TALM を使用してプラットフォームの更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。

- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- 目的のイメージ リポジトリーをミラーリングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新の PolicyGenerator CR を作成します。
   a. 以下の PolicyGenerator CR を du-upgrade.yaml ファイルに保存します。

プラットフォーム更新の PolicyGenerator の例

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
```

```
      noncompliant: 10s
policies:
  - name: du-upgrade-platform-upgrade
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "100"
    manifests:
      - path: source-crs/ClusterVersion.yaml (1)
        patches:
          - metadata:
              name: version
            spec:
              channel: stable-4.16
              desiredUpdate:
                version: 4.16.4
              upstream: http://upgrade.example.com/images/upgrade-graph_stable-
4.16
            status:
              history:
                - state: Completed
                  version: 4.16.4
  - name: du-upgrade-platform-upgrade-prep
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/ImageSignature.yaml (2)
      - path: source-crs/DisconnectedICSP.yaml
        patches:
          - metadata:
              name: disconnected-internal-icsp-for-ocp
            spec:
              repositoryDigestMirrors: (3)
                - mirrors:
                    - quay-intern.example.com/ocp4/openshift-release-dev
                  source: quay.io/openshift-release-dev/ocp-release
                - mirrors:
                    - quay-intern.example.com/ocp4/openshift-release-dev
                  source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

(1)

更新をトリガーする ClusterVersion CR を示します。イメージの事前キャッシュには、channel、upstream、および desiredVersion フィールドがすべて必要です。

(2)

ImageSignature.yaml には、必要なリリースイメージのイメージ署名が含まれます。イメージ署名は、プラットフォームの更新を適用する前に、イメージの検証に使用されます。

3

目的の OpenShift Container Platform イメージを含むミラーリポジトリーを表示します。環境のセットアップセクションの手順に従って保存した imageContentSources.yaml ファイルからミラーを取得します。

PolicyGenerator CR は 2 つのポリシーを生成します。

- du-upgrade-platform-upgrade-prep ポリシーは、プラットフォームの更新の準備作業を行います。目的のリリースイメージシグネチャーの ConfigMap CR を作成し、ミラー化されたリリースイメージリポジトリーのイメージ コンテンツソースを作成し、目的の更新チャネルと切断された環境でマネージドクラスターが到達可能な更新グラフを使用してクラスターバージョンを更新します。
- du-upgrade-platform-upgrade ポリシーは、プラットフォームのアップグレードを実行するために使用されます。

b. PolicyGenTemplate CR の GitOps ZTP Git リポジトリーにある kustomization.yaml ファイルに du-upgrade.yaml ファイルの内容を追加し、変更を Git リポジトリーにプッシュします。

ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

c. 以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep platform-upgrade
```

2. spec.enable フィールドを false に設定して、プラットフォーム更新用の ClusterGroupUpdate CR を作成します。

a. 次の例に示すように、プラットフォーム更新 ClusterGroupUpdate CR の内容を、du-upgrade-platform-upgrade-prep ポリシーと du-upgrade-platform-upgrade ポリシーおよびターゲットクラスターとともに、cgu-platform-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
```

```
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

b. 次のコマンドを実行して、ClusterGroupUpdate CR をハブクラスターに適用します。

```
$ oc apply -f cgu-platform-upgrade.yml
```

3. オプション: プラットフォームの更新用にイメージを事前キャッシュします。

a. 次のコマンドを実行して、ClusterGroupUpdate CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
platform-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 更新プロセスを監視し、事前キャッシュが完了するまで待ちます。ハブクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

4. プラットフォームの更新を開始します。

a. 次のコマンドを実行して、cgu-platform-upgrade ポリシーを有効にし、事前キャッシュを無効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
platform-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠して

いることを確認します。

```
$ oc get policies --all-namespaces
```

関連情報

- 非接続環境でのイメージのミラーリングに関する詳細は、非接続環境の準備 を参照してください。

9.3.3. PolicyGenerator CR を使用した Operator 更新の実行

TALM で Operator の更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- 目的のインデックスイメージ、バンドルイメージ、およびバンドルイメージで参照されるすべての Operator イメージをミラーリングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. Operator 更新の PolicyGenerator CR を更新します。
   a. du-upgrade.yaml ファイルの次の追加コンテンツで du-upgradePolicyGenTemplate CR を更新します。

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno
          operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-operator-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
          - metadata:
              name: redhat-operators
            spec:
              displayName: Red Hat Operators Catalog
              image: registry.example.com:5000/olm/redhat-operators:v4.16 (1)
              updateStrategy: (2)
                registryPoll:
                  interval: 1h
            status:
              connectionState:
                lastObservedState: READY (3)
```

(1) 必要な Operator イメージが含まれている。インデックスイメージが常に同じイメージ名とタグにプッシュされている場合、この変更は必要ありません。

(2) Operator Lifecycle Manager (OLM) が新しい Operator バージョンのインデックスイメージをポーリングする頻度を registryPoll.interval フィールドで設定します。y-stream および z-stream Operator の更新のために新しいインデックスイメージタグが常にプッシュされる場合、この変更は必要ありません。registryPoll.interval フィールドを短い間隔に設定して更新を促進できますが、間隔を短くすると計算負荷

が増加します。これに対処するために、更新が完了したら、registryPoll.interval をデフォルト値に戻すことができます。

3

カタログ接続の監視された状態を表示します。READY 値は、CatalogSource ポリシーの準備が整っていることを保証し、インデックス Pod がプルされ、実行中であることを示します。このように、TALM は最新のポリシー準拠状態に基づいて Operator をアップグレードします。

b.

この更新により、1 つのポリシー du-upgrade-operator-catsrc-policy が生成され、必要な Operator イメージを含む新しいインデックスイメージで redhat-operators カタログソースが更新されます。

注記

Operator にイメージの事前キャッシュを使用する必要があり、redhat-operators 以外の別のカタログソースからの Operator がある場合は、次のタスクを実行する必要があります。

- 別のカタログソースの新しいインデックスイメージまたはレジストリーポーリング間隔の更新を使用して、別のカタログソースポリシーを準備します。
- 異なるカタログソースからの目的の Operator に対して個別のサブスクリプションポリシーを準備します。

たとえば、目的の SRIOV-FEC Operator は、certified-operators カタログソースで入手できます。カタログソースと Operator サブスクリプションを更新するには、次の内容を追加して、2 つのポリシー du-upgrade-fec-catsrc-policy と du-upgrade-subscriptions-fec-policy を生成します。

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: du-upgrade
placementBindingDefaults:
  name: du-upgrade-placement-binding
policyDefaults:
  namespace: ztp-group-du-sno
  placement:
    labelSelector:
```

```
        matchExpressions:
          - key: group-du-sno
            operator: Exists
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
  - name: du-upgrade-fec-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
          - metadata:
              name: certified-operators
            spec:
              displayName: Intel SRIOV-FEC Operator
              image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
              updateStrategy:
                registryPoll:
                  interval: 10m
  - name: du-upgrade-subscriptions-fec-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "2"
    manifests:
      - path: source-crs/AcceleratorsSubscription.yaml
        patches:
          - spec:
              channel: stable
              source: certified-operators
```

c. 共通の PolicyGenerator CR に指定されたサブスクリプションチャネルが存在する場合は、削除します。GItOps ZTP イメージのデフォルトサブスクリプションチャネルが更新に使用されます。

注記

GItOps ZTP 4.15 で適用される Operator のデフォルトチャネルは、performance-addon-operator を除きすべて stable です。OpenShift Container Platform 4.11 以降、performance-addon-operator 機能は node-tuning-operator に移動されました。4.10 リリースの場合、PAO のデフォルトチャネルは v4.10 です。共通の PolicyGenerator CR でデフォルトのチャネルを指定することもできます。

d. PolicyGenerator CR の更新を GitOps ZTP Git リポジトリーにプッシュします。

ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

e. 以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. Operator の更新を開始する前に、必要なカタログソースの更新を適用します。

a. operator-upgrade-prep という名前の ClusterGroupUpgrade CR の内容をカタログソースポリシーと共に、ターゲットマネージドクラスターの内容を cgu-operator-upgrade-prep.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

b. 次のコマンドを実行して、ポリシーをハブ クラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

c. 更新プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. spec.enable フィールドを false に設定して、Operator 更新の ClusterGroupUpgrade CR を作成します。

a. 以下の例のように、Operator 更新 ClusterGroupUpgrade CR の内容を du-upgrade-operator-catsrc-policy ポリシーで保存して、共通の PolicyGenTemplate およびターゲットクラスターで作成されたサブスクリプションポリシーを cgu-operator-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy (1)
  - common-subscriptions-policy (2)
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

(1) このポリシーは、カタログソースから Operator イメージを取得するために、イメージの事前キャッシュ機能で必要になります。

(2) ポリシーには Operator サブスクリプションが含まれます。参照 PolicyGenTemplates の構造と内容に従っている場合、すべての Operator サブスクリプションは common-subscriptions-policy ポリシーにグループ化されます。

注記

1 つの ClusterGroupUpgrade CR は、ClusterGroupUpgrade CR に含まれる 1 つのカタログソースからサブスクリプションポリシーで定義される必要な Operator のイメージのみを事前キャッシュできます。SRIOV-FEC Operator の例のように、目的の Operator が異なるカタログソースからのものである場合、別の ClusterGroupUpgrade CR を du-upgrade-fec-catsrc-policy および du-upgrade-subscriptions-fec-policy ポリシーで作成する必要があります。SRIOV-FEC Operator イメージの事前キャッシュと更新。

b. 次のコマンドを実行して、ClusterGroupUpgrade CR をハブクラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. **オプション: Operator の更新用にイメージを事前キャッシュします。**

a. **イメージの事前キャッシュを開始する前に、以下のコマンドを実行して、サブスクリプションポリシーがこの時点で NonCompliant であることを確認します。**

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

出力例

```
NAME                          REMEDIATION ACTION   COMPLIANCE STATE     AGE
common-subscriptions-policy   inform               NonCompliant         27d
```

b. **以下のコマンドを実行して、ClusterGroupUpgrade CR で事前キャッシュを有効にします。**

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

c. **プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。**

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

d. **以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうかを確認します。**

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
```

出力例

```
[
  {
    "lastTransitionTime": "2022-03-08T20:49:08.000Z",
```

```
      "message": "The ClusterGroupUpgrade CR is not enabled",
      "reason": "UpgradeNotStarted",
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-03-08T20:55:30.000Z",
      "message": "Precaching is completed",
      "reason": "PrecachingCompleted",
      "status": "True",
      "type": "PrecachingDone"
    }
]
```


5. Operator の更新を開始します。

   a. 以下のコマンドを実行して cgu-operator-upgrade ClusterGroupUpgrade CR を有効にし、事前キャッシュを無効にして Operator の更新を開始します。

   ```
   $ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
   --patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
   ```

   b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

   ```
   $ oc get policies --all-namespaces
   ```

関連情報

- GitOps ZTP の更新に関する詳細は、GitOps ZTP のアップグレード を参照してください。

9.3.4. PolicyGenerator CR を使用した失敗した Operator 更新のトラブルシューティング

一部のシナリオでは、ポリシーのコンプライアンス状態が古いため、Topology Aware Lifecycle Manager (TALM) が Operator の更新を見逃す可能性があります。

カタログソースの更新後に Operator Lifecycle Manager (OLM) がサブスクリプションステータスを更新すると、時間がかかります。TALM が修復が必要かどうかを判断する間、サブスクリプションポリ

シーのステータスは準拠していると表示される場合があります。その結果、サブスクリプションポリシーで指定された Operator はアップグレードされません。

このシナリオを回避するには、別のカタログソース設定を PolicyGenTemplate に追加し、更新が必要な Operator のサブスクリプションでこの設定を指定します。

手順

1. PolicyGenerator リソースにカタログソース設定を追加します。

```
manifests:
- path: source-crs/DefaultCatsrc.yaml
  patches:
    - metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v{product-version}
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY
- path: source-crs/DefaultCatsrc.yaml
  patches:
    - metadata:
        name: redhat-operators-v2 (1)
      spec:
        displayName: Red Hat Operators Catalog v2 (2)
        image: registry.example.com:5000/olredhat-operators:<version> (3)
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY
```

(1) 新しい設定の名前を更新します。

(2) 新しい設定の表示名を更新します。

(3)

2. 更新が必要な Operator の新しい設定を指すように Subscription リソースを更新します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: operator-subscription
  namespace: operator-namspace
# ...
spec:
  source: redhat-operators-v2 (1)
# ...
```

(1) PolicyGenTemplate リソースで定義した追加のカタログソース設定の名前を入力します。

9.3.5. プラットフォームと Operator の更新を一緒に実行する

プラットフォームと Operator の更新を同時に実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1.

プラットフォーム更新の実行および Operator 更新の実行セクションで説明されている手順に従って、更新用の PolicyGenTemplate CR を作成します。

2. プラットフォームの準備作業と Operator の更新を適用します。

   a. プラットフォームの更新の準備作業、カタログ ソースの更新、およびターゲット クラスターのポリシーを 含む ClusterGroupUpgrade CR の内容を cgu-platform-operator-upgrade-prep.yml ファイルに保存します。次に例を示します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

   b. 次のコマンドを実行して、cgu-platform-operator-upgrade-prep.yml ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

   c. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

3. プラットフォーム用の ClusterGroupUpdate CR と、spec.enable フィールドを false に設定した Operator 更新を作成します。

   a. 次の例に示すように、ポリシーとターゲットクラスターを含むプラットフォームと Operator の更新 ClusterGroupUpdate CR の内容を cgu-platform-operator-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
```

```
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade 1
  - du-upgrade-operator-catsrc-policy 2
  - common-subscriptions-policy 3
  preCaching: true
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1 これはプラットフォーム更新ポリシーです。

2 これは、更新される Operator のカタログソース情報が含まれるポリシーです。事前キャッシュ機能がマネージドクラスターにダウンロードする Operator イメージを決定するために必要です。

3 これは、Operator を更新するためのポリシーです。

b. 次のコマンドを実行して、cgu-platform-operator-upgrade.yml ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. オプション: プラットフォームおよび Operator の更新用にイメージを事前キャッシュします。

a. 以下のコマンドを実行して、ClusterGroupUpgrade CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b.
更新プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

c.
以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうかを確認します。

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5.
プラットフォームおよび Operator の更新を開始します。

a.
以下のコマンドを実行して、cgu-du-upgrade ClusterGroupUpgrade CR がプラットフォームと Operator の更新を開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b.
プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

注記

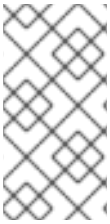
プラットフォームおよび Operator 更新の CR は、設定を spec.enable: true に設定して最初から作成できます。この場合、更新は事前キャッシュが完了した直後に開始し、CR を手動で有効にする必要はありません。

事前キャッシュと更新の両方で、ポリシー、配置バインディング、配置ルール、マネージドクラスターアクション、マネージドクラスタービューなどの追加リソースが作成され、手順を完了することができます。afterCompletion.deleteObjects フィールドを true に設定すると、更新の完了後にこれらのリソースがすべて削除されます。

9.3.6. PolicyGenerator CR を使用してデプロイされたクラスターから Performance Addon Operator サブスクリプションを削除する

以前のバージョンの OpenShift Container Platform では、Performance Addon Operator はアプリケーションの自動低レイテンシーパフォーマンスチューニングを提供していました。OpenShift Container Platform 4.11 以降では、これらの機能は Node Tuning Operator の一部です。

OpenShift Container Platform 4.11 以降を実行しているクラスターに Performance Addon Operator をインストールしないでください。OpenShift Container Platform 4.11 以降にアップグレードすると、Node Tuning Operator は Performance Addon Operator を自動的に削除します。



注記

Operator の再インストールを防ぐために、Performance Addon Operator サブスクリプションを作成するポリシーを削除する必要があります。

参照 DU プロファイルには、PolicyGenerator CR acm-common-ranGen.yaml に Performance Addon Operator が含まれています。デプロイされたマネージドクラスターからサブスクリプションを削除するには、common-ranGen.yaml を更新する必要があります。

注記

Performance Addon Operator 4.10.3-5 以降を OpenShift Container Platform 4.11 以降にインストールする場合、Performance Addon Operator はクラスターのバージョンを検出し、Node Tuning Operator 機能との干渉を避けるために自動的に休止状態になります。ただし、最高のパフォーマンスを確保するには、OpenShift Container Platform 4.11 クラスターから Performance Addon Operator を削除してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。
- OpenShift Container Platform 4.11 以降に更新します。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. common-ranGen.yaml ファイル の Performance Addon Operator namespace、

Operator グループ、およびサブスクリプションの ComplianceType を mustnothave に変更します。

```
- name: group-du-sno-pg-subscriptions-policy
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "2"
  manifests:
    - path: source-crs/PaoSubscriptionNS.yaml
    - path: source-crs/PaoSubscriptionOperGroup.yaml
    - path: source-crs/PaoSubscription.yaml
```

2. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。common-subscriptions-policy ポリシーのステータスが Non-Compliant に変わります。

3. Topology Aware Lifecycle Manager を使用して、ターゲットクラスターに変更を適用します。設定変更のロールアウトの詳細については、「関連情報」セクションを参照してください。

4. プロセスを監視します。ターゲットクラスターの common-subscriptions-policy ポリシーのステータスが Compliant の場合、Performance Addon Operator はクラスターから削除されています。次のコマンドを実行して、common-subscriptions-policy のステータスを取得します。

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

5. acm-common-ranGen.yaml ファイルの policies.manifests から Performance Addon Operator namespace、Operator グループ、およびサブスクリプション CR を削除します。

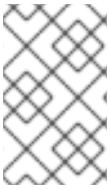
6. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。ポリシーは準拠したままです。

9.3.7. シングルノード OpenShift クラスター上の TALM を使用したユーザー指定のイメージの事前キャッシュ

アプリケーションをアップグレードする前に、アプリケーション固有のワークロードイメージを単一ノード OpenShift クラスターに事前キャッシュできます。

次のカスタムリソース (CR) を使用して、事前キャッシュジョブの設定オプションを指定できます。

- PreCachingConfig CR
- ClusterGroupUpgrade CR



注記

PreCachingConfig CR のフィールドはすべてオプションです。

PreCachingConfig CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: exampleconfig-ns
spec:
  overrides: 1
    platformImage: quay.io/openshift-release-dev/ocp-release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    operatorsIndexes:
      - registry.example.com:5000/custom-redhat-operators:1.0.0
    operatorsPackagesAndChannels:
      - local-storage-operator: stable
      - ptp-operator: stable
      - sriov-network-operator: stable
  spaceRequired: 30 Gi 2
  excludePrecachePatterns: 3
    - aws
    - vsphere
  additionalImages: 4
    - quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    - quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
    - quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
```

1

2

3

イメージ名の一致に基づいて事前キャッシュから除外するイメージを指定します。

4

事前キャッシュする追加イメージのリストを指定します。

PreCachingConfig CR 参照を使用した ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
spec:
  preCaching: true 1
  preCachingConfigRef:
    name: exampleconfig 2
    namespace: exampleconfig-ns 3
```

1

preCaching フィールドを true に設定すると、事前キャッシュジョブが有効になります。

2

preCachingConfigRef.name フィールドは、使用する PreCachingConfig CR を指定します。

3

preCachingConfigRef.namespace は、使用する PreCachingConfig CR の namespace を指定します。

9.3.7.1. 事前キャッシュ用のカスタムリソースの作成

PreCachingConfig CR は、ClusterGroupUpgrade CR の前または同時に作成する必要があります。

1. 事前キャッシュする追加イメージのリストを使用して PreCachingConfig CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: default (1)
spec:
[...]
  spaceRequired: 30Gi (2)
  additionalImages:
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
```

(1) namespace は、ハブクラスターにアクセスできる必要があります。

(2) 事前にキャッシュされたイメージ用に十分なストレージ領域を確保するために、必要な最小ディスク領域フィールドを設定することを推奨します。

2. preCaching フィールドを true に設定して ClusterGroupUpgrade CR を作成し、前の手順で作成した PreCachingConfig CR を指定します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
  namespace: default
spec:
  clusters:
  - sno1
  - sno2
  preCaching: true
  preCachingConfigRef:
```

```
  - name: exampleconfig
    namespace: default
managedPolicies:
  - du-upgrade-platform-upgrade
  - du-upgrade-operator-catsrc-policy
  - common-subscriptions-policy
remediationStrategy:
  timeout: 240
```

警告

クラスターにイメージをインストールすると、それらを変更したり削除したりすることはできません。

3. イメージを事前キャッシュを開始する場合は、次のコマンドを実行して ClusterGroupUpgrade CR CR を適用します。

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

TALM は ClusterGroupUpgrade CR を検証します。

この時点から、TALM 事前キャッシュワークフローを続行できます。

注記

すべてのサイトが同時に事前キャッシュされます。

検証

1. 次のコマンドを実行して、ClusterUpgradeGroup CR が適用されているハブクラスターの事前キャッシュステータスを確認します。

```
$ oc get cgu <cgu_name> -n <cgu_namespace> -oyaml
```

出力例

```
  precaching:
    spec:
      platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2
e1ef
      operatorsIndexes:
        - registry.example.com:5000/custom-redhat-operators:1.0.0
      operatorsPackagesAndChannels:
        - local-storage-operator: stable
        - ptp-operator: stable
        - sriov-network-operator: stable
      excludePrecachePatterns:
        - aws
        - vsphere
      additionalImages:
        -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47e2e1ef
        -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47adfaef
        -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340
adfafdedga0991234afdadfsa09
      spaceRequired: "30"
    status:
      sno1: Starting
      sno2: Starting
```

事前キャッシュ設定は、管理ポリシーが存在するかどうかをチェックすることによって検証されます。ClusterGroupUpgrade および PreCachingConfig CR の設定が有効であると、次のステータスになります。

有効な CR の出力例

```
- lastTransitionTime: "2023-01-01T00:00:01Z"
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: "True"
  type: ClusterSelected
- lastTransitionTime: "2023-01-01T00:00:02Z"
  message: Completed validation
  reason: ValidationCompleted
  status: "True"
```

```
    type: Validated
  - lastTransitionTime: "2023-01-01T00:00:03Z"
    message: Precaching spec is valid and consistent
    reason: PrecacheSpecIsWellFormed
    status: "True"
    type: PrecacheSpecValid
  - lastTransitionTime: "2023-01-01T00:00:04Z"
    message: Precaching in progress for 1 clusters
    reason: InProgress
    status: "False"
    type: PrecachingSucceeded
```

無効な PreCachingConfig CR の例

```
Type:    "PrecacheSpecValid"
Status:  False,
Reason:  "PrecacheSpecIncomplete"
Message: "Precaching spec is incomplete: failed to get PreCachingConfig resource
due to PreCachingConfig.ran.openshift.io "<pre-caching_cr_name>" not found"
```

2. マネージドクラスターで次のコマンドを実行すると、事前キャッシュジョブを見つけることができます。

```
$ oc get jobs -n openshift-talo-pre-cache
```

進行中の事前キャッシュジョブの例

```
NAME        COMPLETIONS      DURATION      AGE
pre-cache   0/1              1s            1s
```

3. 次のコマンドを実行して、事前キャッシュジョブ用に作成された Pod のステータスを確認できます。

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

進行中の事前キャッシュジョブの例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate     19s    job-controller    Created pod: pre-cache-abcd1
```

4. 次のコマンドを実行すると、ジョブのステータスに関するライブ更新を取得できます。

```
$ oc logs -f pre-cache-abcd1 -n openshift-talo-pre-cache
```

5. 事前キャッシュジョブが正常に完了したことを確認するには、次のコマンドを実行します。

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

完了した事前キャッシュジョブの例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate     5m19s  job-controller    Created pod: pre-cache-abcd1
Normal     Completed           19s    job-controller    Job completed
```

6. イメージが単一ノード OpenShift で正常に事前キャッシュ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. デバッグモードでノードに入ります。

   ```
   $ oc debug node/cnfdf00.example.lab
   ```

   b. root を host に変更します。

   -

```
$ chroot /host/
```

c. 目的のイメージを検索します。

```
$ sudo podman images | grep <operator_name>
```

関連情報

- TALM の事前キャッシュワークフローについて、詳細は コンテナーイメージ事前キャッシュ機能の使用 を参照してください。

9.3.8. GitOps ZTP 用に自動作成された ClusterGroupUpgrade CR について

TALM には、ManagedClusterForCGU と呼ばれるコントローラーがあります。このコントローラーは、ハブクラスター上で ManagedCluster CR の Ready 状態を監視し、GitOps Zero Touch Provisioning (ZTP) の ClusterGroupUpgrade CR を作成します。

ztp-done ラベルが適用されていない Ready 状態のマネージドクラスターの場合、ManagedClusterForCGU コントローラーは、ztp-install namespace に ClusterGroupUpgrade CR と、GItOps ZTP プロセス中に作成された関連する RHACM ポリシーを自動的に作成します。次に TALM は自動作成された ClusterGroupUpgrade CR に一覧表示されている設定ポリシーのセットを修正し、設定 CR をマネージドクラスターにプッシュします。

クラスターが Ready になった時点でマネージドクラスターのポリシーがない場合、ポリシーのない ClusterGroupUpgrade CR が作成されます。ClusterGroupUpgrade が完了すると、マネージドクラスターには ztp-done というラベルが付けられます。そのマネージドクラスターに適用するポリシーがある場合は、2 日目の操作として ClusterGroupUpgrade を 手動で作成します。

GitOps ZTP 用に自動作成された ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
```

```
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" (1)
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" (2)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

(1) TALM がクラスター設定を完了する際にマネージドクラスターに適用されます。

(2) TALM が設定ポリシーのデプロイを開始するときにマネージドクラスターに適用されます。

第10章 POLICYGENTEMPLATE リソースを使用したクラスターポリシーの管理

10.1. POLICYGENTEMPLATE リソースを使用したマネージドクラスターポリシーの設定

適用されたポリシーのカスタムリソース (CR) は、プロビジョニングするマネージドクラスターを設定します。Red Hat Advanced Cluster Management (RHACM) が PolicyGenTemplate CR を使用して、適用されるポリシー CR を生成する方法をカスタマイズできます。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および PolicyGenerator CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

10.1.1. PolicyGenTemplate CRD について

PolicyGenTemplate カスタムリソース定義 (CRD) は、PolicyGen ポリシージェネレーターに、どのカスタムリソース (CR) をクラスター設定に含めるか、CR を生成されたポリシーに結合する方法、およびこれらの CR 内のどのアイテムをオーバーレイコンテンツで更新する必要があるかを伝えます。

次の例は、ztp-site-generate 参照コンテナーから抽出された PolicyGenTemplate CR (common-du-ranGen.yaml) を示しています。common-du-ranGen.yaml ファイルは、2 つの Red Hat Advanced Cluster Management (RHACM) ポリシーを定義します。ポリシーは、CR 内の policyName の一意の値ごとに 1 つずつ、設定 CR のコレクションを管理します。common-du-ranGen.yaml は、単一の配置バインディングと配置ルールを作成して、bindingRules セクションにリストされているラベルに基づいてポリシーをクラスターにバインドします。

Example PolicyGenTemplate CR - common-ranGen.yaml

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true" (1)
    du-profile: "latest"
  sourceFiles: (2)
    - fileName: SriovSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogNS.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageNS.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: DefaultCatsrc.yaml (3)
      policyName: "config-policy" (4)
      metadata:
        name: redhat-operators-disconnected
      spec:
        displayName: disconnected-redhat-operators
        image: registry.example.com:5000/disconnected-redhat-operators/disconnected-redhat-
operator-index:v4.9
    - fileName: DisconnectedICSP.yaml
      policyName: "config-policy"
```

```
    spec:
      repositoryDigestMirrors:
      - mirrors:
        - registry.example.com:5000
        source: registry.redhat.io
```

1 common: true は、このラベルを持つすべてのクラスターにポリシーを適用します。

2 sourceFiles の下にリストされているファイルは、インストールされたクラスターの Operator ポリシーを作成します。

3 DefaultCatsrc.yaml は、切断されたレジストリーのカタログソースを設定します。

4 policyName: "config-policy" は、Operator サブスクリプションを設定します。OperatorHub CR はデフォルトを無効にし、この CR は redhat-operators を切断されたレジストリーを指す CatalogSource CR に置き換えます。

PolicyGenTemplate CR は、任意の数の組み込み CR で設定できます。次の例の CR をハブクラスターに適用して、単一の CR を含むポリシーを生成します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno"
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
    - fileName: PtpConfigSlave.yaml
      policyName: "config-policy"
      metadata:
        name: "du-ptp-slave"
      spec:
        profile:
        - name: "slave"
```

```
        interface: "ens5f0"
        ptp4lOpts: "-2 -s --summary_interval -4"
        phc2sysOpts: "-a -r -n 24"
```

ソースファイル PtpConfigSlave.yaml を例として使用すると、ファイルは PtpConfig CR を定義します。PtpConfigSlave サンプルの生成ポリシーは group-du-sno-config-policy という名前です。生成された group-du-sno-config-policy に定義される PtpConfig CR は du-ptp-slave という名前です。PtpConfigSlave.yaml で定義された spec は、du-ptp-slave の下に、ソースファイルで定義された他の spec 項目と共に配置されます。

次の例は、group-du-sno-config-policy CR を示しています。

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: group-du-ptp-config-policy
  namespace: groups-sub
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
spec:
    remediationAction: inform
    disabled: false
    policy-templates:
        - objectDefinition:
            apiVersion: policy.open-cluster-management.io/v1
            kind: ConfigurationPolicy
            metadata:
                name: group-du-ptp-config-policy-config
            spec:
                remediationAction: inform
                severity: low
                namespaceselector:
                    exclude:
                        - kube-*
                    include:
                        - '*'
                object-templates:
                    - complianceType: musthave
                      objectDefinition:
                        apiVersion: ptp.openshift.io/v1
                        kind: PtpConfig
                        metadata:
                            name: du-ptp-slave
                            namespace: openshift-ptp
                        spec:
                            recommend:
                                - match:
                                - nodeLabel: node-role.kubernetes.io/worker-du
                                  priority: 4
                                  profile: slave
```

```
      profile:
        - interface: ens5f0
          name: slave
          phc2sysOpts: -a -r -n 24
          ptp4lConf: |
            [global]
            #
            # Default Data Set
            #
            twoStepFlag 1
            slaveOnly 0
            priority1 128
            priority2 128
            domainNumber 24
```

10.1.2. PolicyGenTemplate CR をカスタマイズする際の推奨事項

サイト設定の PolicyGenTemplate カスタムリソース (CR) をカスタマイズするときは、次のベストプラクティスを考慮してください。

- 必要な数のポリシーを使用します。使用するポリシーが少ないほど、必要なリソースが少なくなります。追加のポリシーごとに、ハブクラスターとデプロイされたマネージドクラスターの CPU 負荷が増大します。CR は PolicyGenTemplate CR の policyName フィールドに基づいてポリシーに統合されます。policyName に同じ値を持つ同じ PolicyGenTemplate の CR は単一のポリシーで管理されます。

- 切断された環境では、すべての Operator を含む単一のインデックスとしてレジストリーを設定することにより、すべての Operator に対して単一のカタログソースを使用します。マネージドクラスターに CatalogSource CR を追加するたびに、CPU 使用率が増加します。

- MachineConfig CR は、インストール時に適用されるように SiteConfig CR に 追加の Manifest として組み込む必要があります。これにより、クラスターがアプリケーションをデプロイする準備ができるまで全体的な時間がかかる可能性があります。

- PolicyGenTemplates は、必要なバージョンを明示的に指定するために channel フィールドを上書きする必要があります。これにより、アップグレード時にソース CR が変更されても、生成されたサブスクリプションが更新されないようになります。

関連情報

- RHACM を使用したクラスターのスケーリングに関する推奨事項は、パフォーマンスおよびスケーラビリティー を参照してください。

注記

ハブクラスターで多数のスポーククラスターを管理する場合は、ポリシーの数を最小限に抑えてリソースの消費を減らします。

複数のコンフィギュレーション CR を 1 つまたは限られた数のポリシーにグループ化することは、ハブクラスター上のポリシーの総数を減らすための 1 つの方法です。サイト設定の管理に共通、グループ、サイトというポリシーの階層を使用する場合は、サイト固有の設定を 1 つのポリシーにまとめることが特に重要である。

10.1.3. RAN デプロイメントの PolicyGenTemplate CR

PolicyGenTemplate (PGT) カスタムリソース (CR) を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してクラスターに適用される設定をカスタマイズします。PolicyGenTemplate CR を使用すると、1 つ以上のポリシーを生成して、クラスターのフリートで設定 CR のセットを管理できます。PGT は、管理された CR のセットを識別し、それらをポリシーにバンドルし、それらの CR をラップするポリシーを構築し、ラベルバインディングルールを使用してポリシーをクラスターに関連付けます。

GitOps ZTP コンテナーから取得した参照設定は、RAN (Radio Access Network) 分散ユニット (DU) アプリケーションに典型的な厳しいパフォーマンスとリソース利用制約をクラスターが確実にサポートできるように、重要な機能とノードのチューニング設定のセットを提供するように設計されています。ベースライン設定の変更または省略は、機能の可用性、パフォーマンス、およびリソースの利用に影響を与える可能性があります。参照 PolicyGenTemplate CR をベースに、お客様のサイト要件に合わせた設定ファイルの階層を作成します。

RAN DU クラスター設定に定義されているベースライン PolicyGenTemplate CR は、GitOps ZTP ztp-site-generate コンテナーから抽出することが可能です。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。

PolicyGenTemplate の CR は、./out/argocd/example/policygentemplates フォルダーに格納されています。参照アーキテクチャーには、common、group、および site 固有の設定 CR があります。各 PolicyGenTemplate CR は ./out/source-crs フォルダーにある他の CR を参照します。

RAN クラスター設定に関連する PolicyGenTemplate CR は以下で説明されています。バリアントは、単一ノード、3 ノードのコンパクト、および標準のクラスター設定の相違点に対応するために、グループ PolicyGenTemplate CR に提供されます。同様に、シングルノードクラスターとマルチノード (コンパクトまたはスタンダード) クラスターについても、サイト固有の設定バリエーションが提供されています。デプロイメントに関連するグループおよびサイト固有の設定バリアントを使用します。

表10.1 RAN デプロイメントの PolicyGenTemplate CR

PolicyGenTemplate CR	説明
example-multinode-site.yaml	マルチノードクラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
example-sno-site.yaml	単一ノードの OpenShift クラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
common-mno-ranGen.yaml	マルチノードクラスターに適用される共通の RAN ポリシー設定のセットが含まれています。
common-ranGen.yaml	すべてのクラスターに適用される共通の RAN CR のセットが含まれています。これらの CR は、RAN の典型的なクラスター機能とベースラインクラスターのチューニングを提供する Operator のセットをサブスクライブします。
group-du-3node-ranGen.yaml	3 ノードクラスター用の RAN ポリシーのみが含まれています。
group-du-sno-ranGen.yaml	シングルノードクラスター用の RAN ポリシーのみが含まれています。
group-du-standard-ranGen.yaml	標準的な 3 つのコントロールプレーンクラスターの RAN ポリシーが含まれています。
group-du-3node-validator-ranGen.yaml	**PolicyGenTemplate** CR は、3 ノードクラスターに必要なさまざまなポリシーを生成するために使用されます。
group-du-standard-validator-ranGen.yaml	標準クラスターに必要なさまざまなポリシーを生成するために使用される **PolicyGenTemplate** CR。
group-du-sno-validator-ranGen.yaml	**PolicyGenTemplate** CR は、単一ノードの OpenShift クラスターに必要なさまざまなポリシーを生成するために使用されます。

関連情報

- GitOps ZTP サイト設定リポジトリーの準備

10.1.4. PolicyGenTemplate CR を使用したマネージドクラスターのカスタマイズ

次の手順を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してプロビジョ

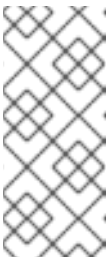
ニングするマネージドクラスターに適用されるポリシーをカスタマイズします。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. サイト固有の設定 CR の PolicyGenTemplate CR を作成します。
   a. CR の適切な例を out/argocd/example/policygentemplates フォルダーから選択します (example-sno-site.yaml または example-multinode-site.yaml)。
   b. サンプルファイルの bindingRules フィールドを、SiteConfig CR に含まれるサイト固有のラベルと一致するように変更します。サンプルの SiteConfig ファイルでは、サイト固有のラベルは sites: example-sno です。

      注記

      PolicyGenTemplate bindingRules フィールドで定義されているラベルが、関連するマネージドクラスターの SiteConfig CR で定義されているラベルに対応していることを確認してください。

   c. サンプルファイルの内容を目的の設定に合わせて変更します。
2. オプション: クラスターのフリート全体に適用される一般的な設定 CR の

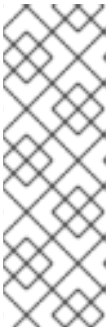
PolicyGenTemplate CR を作成します。

a. out/argocd/example/policygentemplates フォルダーから CR の適切な例を選択します (例: common-ranGen.yaml)。

b. サンプルファイルの内容を目的の設定に合わせて変更します。

3. オプション: フリート内のクラスターの特定のグループに適用されるグループ設定 CR の PolicyGenTemplate CR を作成します。

オーバーレイド仕様ファイルの内容が必要な終了状態と一致することを確認します。out/source-crs ディレクトリーには、PolicyGenTemplate テンプレートに含めることができる source-crs の完全な一覧が含まれます。



注記

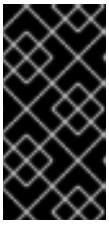
クラスターの特定の要件に応じて、クラスターの種類ごとに 1 つ以上のグループポリシーが必要になる場合があります。特に、サンプルのグループポリシーにはそれぞれ単一の PerformancePolicy.yaml ファイルがあり、それらのクラスターが同一のハードウェア設定である場合にのみクラスターのセット全体で共有できることを考慮しています。

a. out/argocd/example/policygentemplates フォルダーから CR の適切な例を選択します (例: group-du-sno-ranGen.yaml)。

b. サンプルファイルの内容を目的の設定に合わせて変更します。

4. オプション: GitOps ZTP のインストールとデプロイされたクラスターの設定が完了したときに通知するバリデータ通知ポリシー PolicyGenTemplate CR を作成します。詳細は、バリデータ通知ポリシーの作成を参照してください。

5. out/argocd/example/policygentemplates/ns.yaml ファイルの例と同様の YAML ファイルで、すべてのポリシーの namespace を定義してください。



重要

Namespace CR を PolicyGenTemplate CR と同じファイルに含めないでください。

6. out/argocd/example/policygentemplates/kustomization.yaml に示されている例と同様に、PolicyGenTemplate CR と Namespace CR をジェネレーターセクションの kustomization.yaml ファイルに追加します。

7. PolicyGenTemplate CR、Namespace CR、および関連する kustomization.yaml ファイルを Git リポジトリーにコミットし、変更をプッシュします。

ArgoCD パイプラインが変更を検出し、マネージドクラスターのデプロイを開始します。SiteConfig CR と PolicyGenTemplate CR に同時に変更をプッシュすることができます。

関連情報

- バリデーターインフォームポリシーを使用した GitOps ZTP クラスターデプロイメントの完了のシグナリング

10.1.5. マネージドクラスターポリシーのデプロイメントの進行状況の監視

ArgoCD パイプラインは、Git の PolicyGenTemplate CR を使用して RHACM ポリシーを生成し、ハブクラスターに同期します。支援されたサービスが OpenShift Container Platform をマネージドクラスターにインストールした後、管理対象クラスターのポリシー Synchronization の進行状況をモニターできます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。

手順

1. Topology Aware Lifecycle Manager (TALM) は、クラスターにバインドされている設定ポリシーを適用します。

クラスターのインストールが完了し、クラスターが Ready になると、ran.openshift.io/ztp-deploy-wave アノテーションで 定義された順序付きポリシーのリストで、このクラスターに対応する ClusterGroupUpgrade CR が TALM により自動的に作成されます。クラスターのポリシーは、ClusterGroupUpgrade CR に記載されている順序で適用されます。

以下のコマンドを使用して、設定ポリシー調整のハイレベルの進捗を監視できます。

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[-1:]}' | jq
```

出力例

```
{
 "lastTransitionTime": "2022-11-09T07:28:09Z",
 "message": "Remediating non-compliant policies",
 "reason": "InProgress",
 "status": "True",
 "type": "Progressing"
}
```

2. RHACM ダッシュボードまたはコマンドラインを使用して、詳細なクラスターポリシーのコンプライアンスステータスを監視できます。

   a. oc を使用してポリシーのコンプライアンスを確認するには、次のコマンドを実行します。

   ```
   $ oc get policies -n $CLUSTER
   ```

   出力例

   ```
   NAME                                REMEDIATION ACTION   COMPLIANCE
   STATE   AGE
   ztp-common.common-config-policy                 inform           Compliant
   3h42m
   ztp-common.common-subscriptions-policy              inform
   ```

```
NonCompliant       3h42m
ztp-group.group-du-sno-config-policy                 inform            NonCompliant
3h42m
ztp-group.group-du-sno-validator-du-policy           inform            NonCompliant
3h42m
ztp-install.example1-common-config-policy-pjz9s      enforce           Compliant
167m
ztp-install.example1-common-subscriptions-policy-zzd9k   enforce
NonCompliant       164m
ztp-site.example1-config-policy                      inform            NonCompliant
3h42m
ztp-site.example1-perf-policy                        inform            NonCompliant
3h42m
```

b. RHACM Web コンソールからポリシーのステータスを確認するには、次のアクションを実行します。

i. ガバナンス → ポリシーの検索 をクリックします。

ii. クラスターポリシーをクリックして、ステータスを確認します。

すべてのクラスターポリシーが準拠すると、クラスターの GitOps ZTP のインストールと設定が完了します。ztp-done ラベルがクラスターに追加されます。

参照設定では、準拠する最終的なポリシーは、*-du-validator-policy ポリシーで定義されたものです。このポリシーは、クラスターに準拠する場合、すべてのクラスター設定、Operator のインストール、および Operator 設定が完了します。

10.1.6. 設定ポリシー CR の生成の検証

ポリシーのカスタムリソース (CR) は、作成元の PolicyGenTemplate と同じネームスペースで生成される。以下のコマンドを使用して示すように、ztp-common、ztp-group、または ztp-site ベースのいずれであるかにかかわらず、PolicyGenTemplate から生成されたすべてのポリシー CR に同じトラブルシューティングフローが適用されます。

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

予想される policy-wraped CR のセットが表示されるはずです。

ポリシーの同期に失敗した場合は、以下のトラブルシューティング手順を使用します。

手順

1. ポリシーの詳細情報を表示するには、次のコマンドを実行します。

```
$ oc describe -n openshift-gitops application policies
```

2. Status: Conditions: の有無を確認し、エラーログを表示します。たとえば、無効な sourceFile エントリーを fileName: に設定すると、以下のようなエラーが発生します。

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or
directory Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit
status 1: exit status 1
    Type:  ComparisonError
```

3. Status: Sync: をチェックします。Status: Conditions:: でログエラーが発生した場合 Status: Sync: に Unknown または Error と表示されます。

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  policies-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:           policies
        Repo URL:       https://git.com/ran-sites/policies/.git
        Target Revision:  master
    Status:             Error
```

4. Red Hat Advanced Cluster Management (RHACM) が ManagedCluster オブジェクトにポリシーが適用されることを認識すると、ポリシー CR オブジェクトがクラスターネームスペースに適用されます。ポリシーがクラスターネームスペースにコピーされたかどうかを確認します。

```
$ oc get policy -n $CLUSTER
```

出力例:

```
NAME                                    REMEDIATION ACTION   COMPLIANCE STATE   AGE
ztp-common.common-config-policy         inform               Compliant          13d
ztp-common.common-subscriptions-policy  inform               Compliant          13d
ztp-group.group-du-sno-config-policy    inform               Compliant          13d
ztp-group.group-du-sno-validator-du-policy  inform           Compliant          13d
ztp-site.example-sno-config-policy      inform               Compliant          13d
```

RHACM は、適用可能なすべてのポリシーをクラスターの namespace にコピーします。コピーされたポリシー名の形式は <policyGenTemplate.Namespace>.<policyGenTemplate.Name>-<policyName> です。

5. クラスター namespace にコピーされないポリシーの配置ルールを確認します。これらのポリシーの PlacementRule の matchSelector、ManagedCluster オブジェクトのラベルと一致する必要があります。

```
$ oc get PlacementRule -n $NS
```

6. PlacementRule 名は、以下のコマンドを使用して、不足しているポリシー (common、group、または site) に適した名前であることに注意してください。

```
$ oc get PlacementRule -n $NS <placement_rule_name> -o yaml
```

- status-decisions にはクラスター名が含まれている必要があります。
- spec の matchSelector の key-value ペアは、マネージドクラスター上のラベルと一致する必要があります。

7. 以下のコマンドを使用して、ManagedCluster オブジェクトのラベルを確認します。

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. 以下のコマンドを使用して、準拠しているポリシーを確認します。

```
$ oc get policy -n $CLUSTER
```

Namespace、OperatorGroup、および Subscription ポリシーが準拠しているが Operator 設定ポリシーが該当しない場合、Operator はマネージドクラスターにインストールされていない可能性があります。このため、スポークに CRD がまだ適用されていないため、Operator 設定ポリシーの適用に失敗します。

10.1.7. ポリシー調整の再開

たとえば、ClusterGroupUpgrade カスタムリソース (CR) がタイムアウトした場合など、予期しないコンプライアンスの問題が発生した場合は、ポリシー調整を再開できます。

手順

1. ClusterGroupUpgrade CR は、管理クラスターの状態が Ready になった後に Topology Aware Lifecycle Manager によって namespace ztp-install に生成されます。

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. 予期せぬ問題が発生し、設定されたタイムアウト (デフォルトは 4 時間) 内にポリシーが苦情にならなかった場合、ClusterGroupUpgrade CR のステータスは UpgradeTimedOut と 表示されます。

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o
jsonpath='{.status.conditions[?(@.type=="Ready")]}'
```

3. UpgradeTimedOut 状態の ClusterGroupUpgrade CR は、1 時間ごとにポリシー照合を自動的に再開します。ポリシーを変更した場合は、既存の ClusterGroupUpgrade CR を削除して再試行をすぐに開始できます。これにより、ポリシーをすぐに調整する新規 ClusterGroupUpgrade CR の自動作成がトリガーされます。

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

ClusterGroupUpgrade CR が UpgradeCompleted のステータスで完了し、管理対象のクラスターに ztp-done ラベルが適用されると、PolicyGenTemplate を使用して追加の設定変更を行うことができ

ます。既存の ClusterGroupUpgrade CR を削除しても、TALM は新規 CR を生成しません。

この時点で、GitOps ZTP はクラスターとの対話を完了しました。それ以降の対話は更新として扱われ、ポリシーの修復のために新しい ClusterGroupUpgrade CR が作成されます。

関連情報

- Topology Aware Lifecycle Manager (TALM) を使用して独自の ClusterGroupUpgrade CR を作成する方法は、ClusterGroupUpgrade CR について を参照してください。

10.1.8. ポリシーを使用して適用済みマネージドクラスター CR を変更する

ポリシーを使用して、マネージドクラスターにデプロイされたカスタムリソース (CR) からコンテンツを削除できます。

PolicyGenTemplate CR から作成されたすべての Policy CR は、complianceType フィールドがデフォルトで musthave に設定されています。マネージドクラスター上の CR には指定されたコンテンツがすべて含まれているため、コンテンツが削除されていない musthave ポリシーは依然として準拠しています。この設定では、CR からコンテンツを削除すると、TALM はポリシーからコンテンツを削除しますが、そのコンテンツはマネージドクラスター上の CR からは削除されません。

complianceType フィールドを Mustonlyhave に設定することで、ポリシーはクラスター上の CR がポリシーで指定されている内容と完全に一致するようにします。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- RHACM を実行しているハブクラスターからマネージドクラスターをデプロイしている。
- ハブクラスターに Topology Aware Lifecycle Manager がインストールされている。

手順

1. 影響を受ける CR から不要になったコンテンツを削除します。この例では、SriovOperatorConfig CR から disableDrain: false 行が削除されました。

CR の例:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    "node-role.kubernetes.io/$mcp": ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
```

2. group-du-sno-ranGen.yaml ファイル内で、影響を受けるポリシーの complianceType を mustonlyhave に変更します。

サンプル YAML

```
- fileName: SriovOperatorConfig.yaml
  policyName: "config-policy"
  complianceType: mustonlyhave
```

3. ClusterGroupUpdates CR を作成し、CR の変更を受け取る必要があるクラスターを指定します。

ClusterGroupUpdates CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
```

```
metadata:
  name: cgu-remove
  namespace: default
spec:
  managedPolicies:
    - ztp-group.group-du-sno-config-policy
  enable: false
  clusters:
  - spoke1
  - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:
```

4. 以下のコマンドを実行して ClusterGroupUpgrade CR を作成します。

```
$ oc create -f cgu-remove.yaml
```

5. たとえば適切なメンテナンス期間中などに変更を適用する準備が完了したら、次のコマンドを実行して spec.enable フィールドの値を true に変更します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

検証

1. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get <kind> <changed_cr_name>
```

出力例

```
NAMESPACE   NAME                                        REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-ztp-group.group-du-sno-config-policy          enforce
17m
default     ztp-group.group-du-sno-config-policy              inform
NonCompliant       15h
```

ポリシーの COMPLIANCE STATE が Compliant の場合、CR が更新され、不要なコンテンツが削除されたことを意味します。

2. マネージドクラスターで次のコマンドを実行して、対象クラスターからポリシーが削除されたことを確認します。

```
$ oc get <kind> <changed_cr_name>
```

結果がない場合、CR はマネージドクラスターから削除されます。

10.1.9. GitOps ZTP インストール完了の表示

GitOps Zero Touch Provisioning (ZTP) は、クラスターの GitOps ZTP インストールステータスを確認するプロセスを単純化します。GitOps ZTP ステータスは、クラスターのインストール、クラスター設定、GitOps ZTP 完了の 3 つのフェーズを遷移します。

クラスターインストールフェーズ

クラスターのインストールフェーズは、ManagedCluster CR の ManagedClusterJoined および ManagedClusterAvailable 条件によって示されます。ManagedCluster CR にこの条件がない場合や、条件が False に設定されている場合、クラスターはインストールフェーズに残ります。インストールに関する追加情報は、AgentClusterInstall および ClusterDeployment CR から入手できます。詳細は、Troubleshooting GitOps ZTP を参照してください。

クラスター設定フェーズ

クラスター設定フェーズは、クラスターの ManagedCluster CR に適用される ztp-running ラベルで示されます。

GitOps ZTP 完了

クラスターのインストールと設定は、GitOps ZTP 完了フェーズで実行されます。これは、ztp-running ラベルを削除し、ManagedCluster CR に ztp-done ラベルを追加することで表示されます。ztp-done ラベルは、設定が適用され、ベースライン DU 設定が完了したことを示しています。

ZTP 完了状態への遷移は、Red Hat Advanced Cluster Management (RHACM) バリデーターのインフォームドポリシーの準拠状態が条件となります。このポリシーは、完了したインストールの既存の基準をキャプチャし、マネージドクラスターの GitOps ZTP プロビジョニングが完了したときにのみ、準拠した状態に移行することを検証するものです。

バリデータ通知ポリシーは、クラスターの設定が完全に適用され、Operator が初期化を完了したことを確認します。ポリシーは以下を検証します。

- ターゲット MachineConfigPool には予想されるエントリーが含まれ、更新が完了しました。全ノードが利用可能で、低下することはありません。
- SR-IOV Operator は、syncStatus: Succeeded の 1 つ以上の SriovNetworkNodeState によって示されているように初期化を完了しています。
- PTP Operator デーモンセットが存在する。

10.2. POLICYGENTEMPLATE リソースを使用した高度なマネージドクラスター設定

PolicyGenTemplate CR を使用して、マネージドクラスターにカスタム機能をデプロイできます。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および PolicyGenerator CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

10.2.1. 追加の変更のクラスターへのデプロイ

基本の GitOps Zero Touch Provisioning (ZTP) パイプライン設定以外のクラスター設定を変更する必要がある場合、次の 3 つのオプションを実行できます。

ZTP パイプラインの完了後に追加設定を適用する

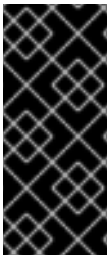
GitOps ZTP パイプラインのデプロイが完了すると、デプロイされたクラスターはアプリケーションのワークロードに対応できるようになります。この時点で、Operator を追加インストールし、お客様の要件に応じた設定を適用することができます。追加のコンフィギュレーションがプラットフォームのパフォーマンスや割り当てられた CPU バジェットに悪影響を与えないことを確認する。

GitOps ZTP ライブラリーにコンテンツを追加する

GitOps ZTP パイプラインでデプロイするベースソースのカスタムリソース (CR) は、必要に応じてカスタムコンテンツで拡張できます。

クラスターインストール用の追加マニフェストの作成

インストール時に余分なマニフェストが適用され、インストール作業を効率化することができます。



重要

追加のソース CR を提供したり、既存のソース CR を変更したりすると、OpenShift Container Platform のパフォーマンスまたは CPU プロファイルに大きな影響を与える可能性があります。

10.2.2. PolicyGenTemplate CR を使用して、ソース CR の内容を上書きする。

PolicyGenTemplate カスタムリソース (CR) を使用すると、ztp-site-generate コンテナーの GitOps プラグインで提供されるベースソース CR の上に追加の設定の詳細をオーバーレイできます。PolicyGenTemplate CR は、ベース CR の論理マージまたはパッチとして解釈できます。PolicyGenTemplate CR を使用して、ベース CR の単一フィールドを更新するか、ベース CR の内容全体をオーバーレイします。ベース CR にない値の更新やフィールドの挿入が可能です。

以下の手順例では、group-du-sno-ranGen.yaml ファイル内の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR のフィールドを更新する方法について説明します。この手順を元に、PolicyGenTemplate の 他の部分をお客様のご要望に応じて変更してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。

手順

1. 既存のコンテンツのベースラインソース CR を確認します。参照 PolicyGenTemplate CR に記載されているソース CR を GitOps Zero Touch Provisioning (ZTP) コンテナーから抽出し、確認すできます。

   a. /out フォルダーを作成します。

   ```
   $ mkdir -p ./out
   ```

   b. ソース CR を抽出します。

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.16.1 extract /home/ztp --tar | tar x -C ./out
   ```

2. ./out/source-crs/PerformanceProfile.yaml にあるベースライン PerformanceProfile CR を確認します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```

注記

ソース CR のフィールドで $... を含むものは、PolicyGenTemplate CR で提供されない場合、生成された CR から削除されます。

3. group-du-sno-ranGen.yaml リファレンスファイルの PerformanceProfile の PolicyGenTemplate エントリーを更新します。次の例の PolicyGenTemplate CR スタンザは、適切な CPU 仕様を提供し、hugepages 設定を設定し、globallyDisableIrqLoadBalancing を false に設定する新しいフィールドを追加しています。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    name: openshift-node-performance-profile
  spec:
    cpu:
      # These must be tailored for the specific hardware platform
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
        - size: 1G
          count: 10
    globallyDisableIrqLoadBalancing: false
```

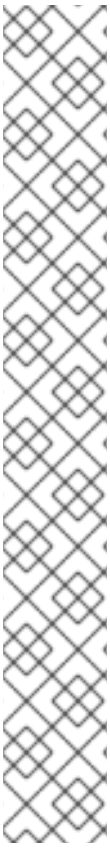
4. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

出力例

GitOps ZTP アプリケーションは、生成された PerformanceProfile CR を含む RHACM ポリシーを生成します。この CR の内容は， PolicyGenTemplate の PerformanceProfile エントリーから metadata と spec の内容をソース CR にマージすることで得られるものである. 作成される CR には以下のコンテンツが含まれます。

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: openshift-node-performance-profile
spec:
    additionalKernelArgs:
        - idle=poll
        - rcupdate.rcu_normal_after_boot=0
    cpu:
```

```
    isolated: 2-19,22-39
    reserved: 0-1,20-21
globallyDisableIrqLoadBalancing: false
hugepages:
  defaultHugepagesSize: 1G
  pages:
    - count: 10
      size: 1G
machineConfigPoolSelector:
  pools.operator.machineconfiguration.openshift.io/master: ""
net:
  userLevelNetworking: true
nodeSelector:
  node-role.kubernetes.io/master: ""
numa:
  topologyPolicy: restricted
realTimeKernel:
  enabled: true
```



注記

ztp-site-generate コンテナーからデプロイメントした /source-crs フォルダーでは、$ 構文が暗示するテンプレート置換は使用されません。むしろ、policyGen ツールが文字列の $ 接頭辞を認識し、関連する PolicyGenTemplate CR でそのフィールドの値を指定しない場合、そのフィールドは出力 CR から完全に省かれます。

例外として、/source-crs YAML ファイル内の $mcp 変数は、PolicyGenTemplate CR から mcp の 指定値で代用されます。例えば、example/policygentemplates/group-du-standard-ranGen.yaml では、mcp の 値は worker となって います。

```
spec:
  bindingRules:
    group-du-standard: ""
  mcp: "worker"
```

policyGen ツールは、$mcp のインスタンスを出力 CR の worker に置き換えます。

10.2.3. GitOps ZTP パイプラインへのカスタムコンテンツの追加

GitOps ZTP パイプラインに新しいコンテンツを追加するには、次の手順を実行します。

手順

1. PolicyGenTemplate カスタムリソース (CR) の kustomization.yaml ファイルが含まれるディレクトリーに、source-crs という名前のサブディレクトリーを作成します。

2. 次の例に示すように、ユーザー提供の CR を source-crs サブディレクトリーに追加します。

```
example
└── policygentemplates
    ├── dev.yaml
    ├── kustomization.yaml
    ├── mec-edge-sno1.yaml
    ├── sno.yaml
    └── source-crs 1
        ├── PaoCatalogSource.yaml
        ├── PaoSubscription.yaml
        ├── custom-crs
        │   ├── apiserver-config.yaml
        │   └── disable-nic-lldp.yaml
        └── elasticsearch
            ├── ElasticsearchNS.yaml
            └── ElasticsearchOperatorGroup.yaml
```

1 source-crs サブディレクトリーは、kustomization.yaml ファイルと同じディレクトリーにある必要があります。

3. 必要な PolicyGenTemplate CR を更新して、source-crs/custom-crs および source-crs/elasticsearch ディレクトリーに追加したコンテンツへの参照を含めます。以下に例を示します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-dev"
  namespace: "ztp-clusters"
spec:
  bindingRules:
    dev: "true"
  mcp: "master"
  sourceFiles:
    # These policies/CRs come from the internal container Image
    #Cluster Logging
    - fileName: ClusterLogNS.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-ns"
    - fileName: ClusterLogOperGroup.yaml
      remediationAction: inform
      policyName: "group-dev-cluster-log-operator-group"
    - fileName: ClusterLogSubscription.yaml
      remediationAction: inform
```

```
    policyName: "group-dev-cluster-log-sub"
  #Local Storage Operator
  - fileName: StorageNS.yaml
    remediationAction: inform
    policyName: "group-dev-lso-ns"
  - fileName: StorageOperGroup.yaml
    remediationAction: inform
    policyName: "group-dev-lso-operator-group"
  - fileName: StorageSubscription.yaml
    remediationAction: inform
    policyName: "group-dev-lso-sub"
  #These are custom local polices that come from the source-crs directory in the git
repo
  # Performance Addon Operator
  - fileName: PaoSubscriptionNS.yaml
    remediationAction: inform
    policyName: "group-dev-pao-ns"
  - fileName: PaoSubscriptionCatalogSource.yaml
    remediationAction: inform
    policyName: "group-dev-pao-cat-source"
    spec:
      image: <container_image_url>
  - fileName: PaoSubscription.yaml
    remediationAction: inform
    policyName: "group-dev-pao-sub"
  #Elasticsearch Operator
  - fileName: elasticsearch/ElasticsearchNS.yaml (1)
    remediationAction: inform
    policyName: "group-dev-elasticsearch-ns"
  - fileName: elasticsearch/ElasticsearchOperatorGroup.yaml
    remediationAction: inform
    policyName: "group-dev-elasticsearch-operator-group"
  #Custom Resources
  - fileName: custom-crs/apiserver-config.yaml (2)
    remediationAction: inform
    policyName: "group-dev-apiserver-config"
  - fileName: custom-crs/disable-nic-lldp.yaml
    remediationAction: inform
    policyName: "group-dev-disable-nic-lldp"
```

(1) (2) /source-crs 親ディレクトリーからのファイルへの相対パスを含むように fileName を設定します。

4. Git で PolicyGenTemplate の変更をコミットし、GitOps ZTP Argo CD ポリシーアプリケーションが監視する Git リポジトリーにプッシュします。

5. ClusterGroupUpgrade CR を更新して、変更された PolicyGenTemplate を含め、cgu-test.yaml として保存します。次の例は、生成された cgu-test.yaml ファイルを示しています。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: custom-source-cr
  namespace: ztp-clusters
spec:
  managedPolicies:
    - group-dev-config-policy
  enable: true
  clusters:
  - cluster1
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

6. 次のコマンドを実行して、更新された ClusterGroupUpgrade CR を適用します。

```
$ oc apply -f cgu-test.yaml
```

検証

- 次のコマンドを実行して、更新が成功したことを確認します。

```
$ oc get cgu -A
```

出力例

```
NAMESPACE     NAME             AGE   STATE       DETAILS
ztp-clusters  custom-source-cr  6s    InProgress   Remediating non-compliant policies
ztp-install   cluster1         19h   Completed    All clusters are compliant with all the
managed policies
```

10.2.4. PolicyGenTemplate CR のポリシーコンプライアンス評価タイムアウトの設定

ハブクラスターにインストールされた Red Hat Advanced Cluster Management (RHACM) を使用して、管理対象クラスターが適用されたポリシーに準拠しているかどうかを監視および報告します。RHACM は、ポリシーテンプレートを使用して、定義済みのポリシーコントローラーとポリシーを適用します。ポリシーコントローラーは Kubernetes のカスタムリソース定義 (CRD) インスタンスです。

デフォルトのポリシー評価間隔は、PolicyGenTemplate カスタムリソース (CR) でオーバーライド

できます。RHACM が適用されたクラスターポリシーを再評価する前に、ConfigurationPolicy CR がポリシー準拠または非準拠の状態を維持できる期間を定義する期間設定を設定します。

GitOps Zero Touch Provisioning (ZTP) ポリシージェネレーターは、事前定義されたポリシー評価間隔で ConfigurationPolicy CR ポリシーを生成します。noncompliant 状態のデフォルト値は 10 秒です。compliant 状態のデフォルト値は 10 分です。評価間隔を無効にするには、値を never に設定します。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. PolicyGenTemplate CR のすべてのポリシーの評価間隔を設定するには、evaluationInterval フィールドに適切な compliant 値と noncompliant 値を設定します。以下に例を示します。

```
spec:
  evaluationInterval:
    compliant: 30m
    noncompliant: 20s
```

注記

また、準拠フィールドと 非 準拠 フィールドを never に設定して、特定のコンプライアンス状態に達した後にポリシーの評価を停止することもできます。

2. PolicyGenTemplate CR で個々のポリシーオブジェクトの評価間隔を設定するには、evaluationInterval フィールドを追加し、適切な値を設定します。以下に例を示します。

```
spec:
  sourceFiles:
    - fileName: SriovSubscription.yaml
      policyName: "sriov-sub-policy"
```

```
    evaluationInterval:
      compliant: never
      noncompliant: 10s
```

3. PolicyGenTemplate CR ファイルを Git リポジトリーにコミットし、変更をプッシュします。

検証

マネージドスポーククラスターポリシーが予想される間隔で監視されていることを確認します。

1. 管理対象クラスターで cluster-admin 権限を持つユーザーとしてログインします。

2. open-cluster-management-agent-addon namespace で実行されている Pod を取得します。以下のコマンドを実行します。

```
$ oc get pods -n open-cluster-management-agent-addon
```

出力例

```
NAME                                READY   STATUS    RESTARTS       AGE
config-policy-controller-858b894c68-v4xdb    1/1     Running   22 (5d8h ago)   10d
```

3. config-policy-controller Pod のログで、適用されたポリシーが予想される間隔で評価されていることを確認します。

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-
858b894c68-v4xdb
```

出力例

```
2022-05-10T15:10:25.280Z       info   configuration-policy-controller
controllers/configurationpolicy_controller.go:166      Skipping the policy evaluation
due to the policy not reaching the evaluation interval  {"policy": "compute-1-config-
policy-config"}
```

```
2022-05-10T15:10:25.280Z info configuration-policy-controller
controllers/configurationpolicy_controller.go:166 Skipping the policy evaluation
due to the policy not reaching the evaluation interval {"policy": "compute-1-common-
compute-1-catalog-policy-config"}
```

10.2.5. バリデーターインフォームポリシーを使用した GitOps ZTP クラスターデプロイメントの完了のシグナリング

デプロイされたクラスターの GitOps Zero Touch Provisioning (ZTP) のインストールと設定が完了したときに通知するバリデーター通知ポリシーを作成します。このポリシーは、単一ノード OpenShift クラスター、3 ノードクラスター、および標準クラスターのデプロイメントに使用できます。

手順

1. ソースファイル validatorCRs/informDuValidator.yaml を含むスタンドアロンの PolicyGenTemplate カスタムリソース (CR) を作成します。スタンドアロン PolicyGenTemplate CR は、各クラスタータイプに 1 つだけ必要です。たとえば、次の CR は、単一ノードの OpenShift クラスターにバリデータ通知ポリシーを適用します。

   単一ノードクラスターバリデータ通知ポリシー CR の例 (group-du-sno-validator-ranGen.yaml)

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-sno: "" 3
  bindingExcludedRules:
    ztp-done: "" 4
  mcp: "master" 5
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform 6
      policyName: "du-policy" 7
```

1

2

この値は、グループ policy-gen-crs で使用される namespace と一致する必要があります。

3

bindingRules で定義された group-du-* ラベルは SiteConfig ファイルに存在している必要があります。

4

bindingExcludedRules で定義されたラベルは 'ztp-done:' でなければなりません。ztp-done ラベルは、Topology Aware Lifecycle Manager と調整するために使用されます。

5

mcp はソースファイル validatorCRs/informDuValidator.yaml で使用される MachineConfigPool オブジェクトを定義する。これは、単一ノードの場合は master であり、標準のクラスターデプロイメントの場合は 3 ノードクラスターデプロイメントおよび worker である必要があります。

6

オプション: デフォルト値は inform です。

7

この値は、生成された RHACM ポリシーの名前の一部として使用されます。単一ノードの例の生成されたバリデーターポリシーは group-du-sno-validator-du-policy という名前です。

2. PolicyGenTemplate CR ファイルを Git リポジトリーにコミットし、変更をプッシュします。

関連情報

- GitOps ZTP のアップグレード

10.2.6. PolicyGenTemplates CR を使用して電源状態を設定する

低レイテンシーで高パフォーマンスのエッジデプロイメントでは、C ステートと P ステートを無効にするか制限する必要があります。この設定では、CPU は一定の周波数 (通常は最大ターボ周波数) で実行されます。これにより、CPU が常に最大速度で実行され、高いパフォーマンスと低レイテンシーが実現されます。これにより、ワークロードのレイテンシーが最適化されます。ただし、これは最大の電力消費にもつながり、すべてのワークロードに必要ではない可能性があります。

ワークロードはクリティカルまたは非クリティカルとして分類できます。クリティカルなワークロードでは、高パフォーマンスと低レイテンシーのために C ステートと P ステートの設定を無効にする必要があります。クリティカルでないワークロードでは、C ステートと P ステートの設定を使用して、いくらかのレイテンシーとパフォーマンスを犠牲にします。GitOps Zero Touch Provisioning (ZTP) を使用して、次の 3 つの電源状態を設定できます。

- 高性能モードは、最大の消費電力で超低遅延を提供します。
- パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。
- 省電力は、消費電力の削減と遅延の増加のバランスをとります。

デフォルトの設定は、低遅延のパフォーマンスモードです。

PolicyGenTemplate カスタムリソース (CR) を使用すると、ztp-site-generate コンテナーの GitOps プラグインで提供されるベースソース CR に追加の設定の詳細をオーバーレイできます。

group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して、電源状態を設定します。

次の共通の前提条件は、3 つの電源状態すべての設定に適用されます。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。
- GitOps ZTP サイト設定リポジトリーの準備で説明されている手順に従っていること。

関連情報

- ワークロードヒントを使用したノードの電力消費とリアルタイム処理の設定

10.2.6.1. PolicyGenTemplate CR を使用してパフォーマンスモードを設定する

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新してパフォーマンスモードを設定します。

パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

1. out/argocd/example/policygentemplates にある group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenTemplate エントリーを次のように更新して、パフォーマンスモードを設定します。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
    # ...
    workloadHints:
      realTime: true
      highPowerConsumption: false
      perPodPowerManagement: false
```

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

10.2.6.2. PolicyGenTemplate CR を使用した高パフォーマンスモードの設定

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して高パフォーマンスモー

ドを設定します。

高パフォーマンスモードは、最大の消費電力で超低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

1. out/argocd/example/policygentemplates にある group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenTemplate エントリーを次のように更新して、高パフォーマンスモードを設定します。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
  # ...
    workloadHints:
        realTime: true
        highPowerConsumption: true
        perPodPowerManagement: false
```

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

10.2.6.3. PolicyGenTemplate CR を使用した省電力モードの設定

この例に従って group-du-sno-ranGen.yaml の PolicyGenTemplate CR に基づいて、参照設定用に生成された PerformanceProfile CR の workloadHints フィールドを更新して、省電力モードを設定します。

省電力モードは、消費電力の削減と遅延の増加のバランスをとります。

前提条件

- BIOS で C ステートと OS 制御の P ステートを有効にしました。

手順

1. out/argocd/example/policygentemplates にある group-du-sno-ranGen.yaml 参照ファイルの PerformanceProfile の PolicyGenTemplate エントリーを次のように更新して、省電力モードを設定します。追加のカーネル引数オブジェクトを使用して、省電力モード用に CPU ガバナーを設定することを推奨します。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
  # ...
  spec:
    # ...
   workloadHints:
     realTime: true
     highPowerConsumption: false
     perPodPowerManagement: true
   # ...
   additionalKernelArgs:
     - # ...
     - "cpufreq.default_governor=schedutil" (1)
```

(1) schedutil ガバナーが推奨されますが、使用できる他のガバナーには ondemand と powersave が含まれます。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

検証

1. 次のコマンドを使用して、識別されたノードのリストから、デプロイされたクラスター内のワーカーノードを選択します。

```
$ oc get nodes
```

2. 次のコマンドを使用して、ノードにログインします。

```
$ oc debug node/<node-name>
```

<node-name> を、電源状態を確認するノードの名前に置き換えます。

3. /host をデバッグシェル内の root ディレクトリーとして設定します。デバッグ Pod は、Pod 内の /host にホストの root ファイルシステムをマウントします。次の例に示すように、ルートディレクトリーを /host に変更すると、ホストの実行可能パスに含まれるバイナリーを実行できます。

```
# chroot /host
```

4. 次のコマンドを実行して、適用された電源状態を確認します。

```
# cat /proc/cmdline
```

予想される出力

- 省電力モードの intel_pstate=passive。

関連情報

- 高優先度のワークロードと低優先度のワークロードを同じ場所で実行するノードの省電力設定
- 低遅延と高パフォーマンスのためのホストファームウェアの設定
- GitOps ZTP サイト設定リポジトリーの準備

10.2.6.4. 省電力の最大化

最大の CPU 周波数を制限して、最大の電力節約を実現することを推奨します。最大 CPU 周波数を制限せずに重要でないワークロード CPU で C ステートを有効にすると、重要な CPU の周波数が高くなるため、消費電力の節約の多くが無効になります。

sysfs プラグインフィールドを更新し、リファレンス設定の TunedPerformancePatch CR で max_perf_pct に適切な値を設定することで、電力の節約を最大化します。group-du-sno-ranGen.yaml に基づくこの例では、最大 CPU 周波数を制限するために従う手順について説明します。

前提条件

- PolicyGenTemplate CR を使用した省電力モードの設定の説明に従って、省電力モードを

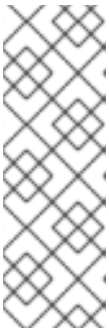
設定しました。

手順

1. out/argocd/example/policygentemplates の group-du-sno-ranGen.yaml 参照ファイルで、TunedPerformancePatch の PolicyGenTemplate エントリーを更新します。電力を最大限に節約するには、次の例に示すように max_perf_pct を追加します。

```
- fileName: TunedPerformancePatch.yaml
  policyName: "config-policy"
  spec:
    profile:
      - name: performance-patch
        data: |
          # ...
          [sysfs]
          /sys/devices/system/cpu/intel_pstate/max_perf_pct=<x> 1
```

1. max_perf_pct は、cpufreq ドライバーが設定できる最大周波数を、サポートされている最大 CPU 周波数のパーセンテージとして制御します。この値はすべての CPU に適用されます。サポートされている最大周波数は /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq で確認できます。開始点として、All Cores Turbo 周波数ですべての CPU を制限する割合を使用できます。All Cores Turbo 周波数は、すべてのコアがすべて使用されているときに全コアが実行される周波数です。



注記

省電力を最大化するには、より低い値を設定します。max_perf_pct の値を低く設定すると、最大 CPU 周波数が制限されるため、消費電力が削減されますが、パフォーマンスに影響を与える可能性もあります。さまざまな値を試し、システムのパフォーマンスと消費電力を監視して、ユースケースに最適な設定を見つけてください。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

10.2.7. PolicyGenTemplate CR を使用した LVM ストレージの設定

GitOps Zero Touch Provisioning (ZTP) を使用して、デプロイするマネージドクラスターの論理ボリュームマネージャー (LVM) ストレージを設定できます。

注記

HTTP トランスポートで PTP イベントまたはベアメタルハードウェアイベントを使用する場合、LVM ストレージを使用してイベントサブスクリプションを永続化します。

分散ユニットでローカルボリュームを使用する永続ストレージには、Local Storage Operator を使用します。

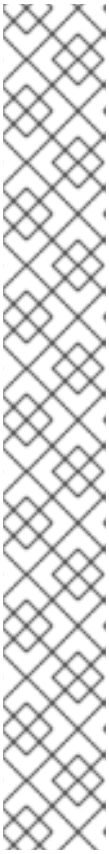
前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成している。

手順

1. 新しいマネージドクラスター用に LVM ストレージを設定するには、次の YAML を common-ranGen.yaml ファイルの spec.sourceFiles に追加します。

```
- fileName: StorageLVMOSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscription.yaml
  spec:
    name: lvms-operator
    channel: stable-4.16
  policyName: subscription-policies
```



注記

Storage LVMO サブスクリプションは非推奨になりました。OpenShift Container Platform の将来のリリースでは、ストレージ LVMO サブスクリプションは利用できなくなります。代わりに、Storage LVMS サブスクリプションを使用する必要があります。

OpenShift Container Platform 4.15 では、LVMO サブスクリプションの代わりに Storage LVMS サブスクリプションを使用できます。LVMS サブスクリプションでは、common-ranGen.yaml ファイルを手動で上書きする必要はありません。Storage LVMS サブスクリプションを使用するには、次の YAML を common-ranGen.yaml ファイルの spec.sourceFiles に追加します。

```
- fileName: StorageLVMSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscription.yaml
  policyName: subscription-policies
```

2. 特定のグループまたは個々のサイト設定ファイルの spec.sourceFiles に LVMCluster CR を追加します。たとえば、group-du-sno-ranGen.yaml ファイルに次を追加します。

```
- fileName: StorageLVMCluster.yaml
  policyName: "lvms-config"
  spec:
    storage:
      deviceClasses:
      - name: vg1
        thinPoolConfig:
          name: thin-pool-1
          sizePercent: 90
          overprovisionRatio: 10
```

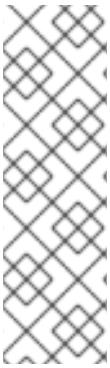
この設定例では、OpenShift Container Platform がインストールされているディスクを除く、使用可能なすべてのデバイスを含むボリュームグループ (vg1) を作成します。シンプール論理ボリュームも作成されます。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

4. Git で PolicyGenTemplate の変更をコミットし、その変更をサイト設定リポジトリーにプッシュして、GitOps ZTP を使用して LVM ストレージを新しいサイトにデプロイします。

10.2.8. PolicyGenTemplate CR を使用した PTP イベントの設定

GitOps ZTP パイプラインを使用して、HTTP または AMQP トランスポートを使用する PTP イベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

10.2.8.1. HTTP トランスポートを使用する PTP イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、HTTP トランスポートを使用する PTP イベントを設定できます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. 要件に応じて、以下の PolicyGenTemplate の変更を group-du-3node-ranGen.yaml、group-du-sno-ranGen.yaml、または group-du-standard-ranGen.yaml ファイルに適用してください。

   a. .sourceFiles に、トランスポートホストを設定する PtpOperatorConfig CR ファイルを追加します。

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
```

```
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: http://ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043
```

注記

OpenShift Container Platform 4.13 以降では、PTP イベントに HTTP トランスポートを使用するときに、PtpOperatorConfig リソースの transportHost フィールドを設定する必要はありません。

b.

PTP クロックの種類とインターフェイスに linuxptp と phc2sys を設定します。たとえば、以下の YAML を spec.sourceFiles に追加します。

```
- fileName: PtpConfigSlave.yaml (1)
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
    - name: "slave"
      interface: "ens5f1" (2)
      ptp4lOpts: "-2 -s --summary_interval -4" (3)
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" (4)
    ptpClockThreshold: (5)
      holdOverTimeout: 30 # seconds
      maxOffsetThreshold: 100  # nano seconds
      minOffsetThreshold: -100
```

(1)

必要に応じて、PtpConfigMaster.yaml、PtpConfigSlave.yaml、または PtpConfigSlaveCvl.yaml のいずれか 1 つを指定できます。PtpConfigSlaveCvl.yaml は、Intel E810 Columbiaville NIC の linuxptp サービスを設定します。group-du-sno-ranGen.yaml および group-du-3node-ranGen.yaml に基づいて設定する場合は、PtpConfigSlave.yaml を使用します。

(2)

デバイス固有のインターフェイス名。

(3)

PTP 高速イベントを有効にするには、.spec.sourceFiles.spec.profile の ptp4lOpts に --summary_interval -4 値を追加する必要があります。

4

phc2sysOpts の値が必要です。-m はメッセージを stdout に出力します。linuxptp-daemon DaemonSet はログを解析し、Prometheus メトリックを生成します。

5

オプション: ptpClockThreshold スタンザが存在しない場合は、ptpClockThreshold フィールドにデフォルト値が使用されます。スタンザは、デフォルトの ptpClockThreshold 値を示します。ptpClockThreshold 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。holdOverTimeout は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が FREERUN に変わるまでの時間値 (秒単位) です。maxOffsetThreshold および minOffsetThreshold 設定は、CLOCK_REALTIME (phc2sys) またはマスターオフセット (ptp4l) の値と比較するナノ秒単位のオフセット値を設定します。ptp4l または phc2sys のオフセット値がこの範囲外の場合、PTP クロックの状態が FREERUN に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が LOCKED に設定されます。

2. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

3. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- PolicyGenTemplate CR を使用して、ソース CR の内容を上書きする。

10.2.8.2. AMQP トランスポートを使用する PTP イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイするマネージドクラスター上で、AMQP トランスポートを使用する PTP イベントを設定できます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. common-ranGen.yaml ファイルの .spec.sourceFiles に以下の YAML を追加し、AMQP Operator を設定します。

```
#AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
```

2. 要件に応じて、以下の PolicyGenTemplate の変更を group-du-3node-ranGen.yaml、group-du-sno-ranGen.yaml、または group-du-standard-ranGen.yaml ファイルに適用してください。

   a. .sourceFiles に、AMQ トランスポートホストを設定する PtpOperatorConfig CR ファイルを config-policy に追加します。

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: "amqp://amq-router.amq-router.svc.cluster.local"
```

   b. PTP クロックの種類とインターフェイスに linuxptp と phc2sys を設定します。たとえば、以下の YAML を spec.sourceFiles に追加します。

```
- fileName: PtpConfigSlave.yaml (1)
  policyName: "config-policy"
```

```
metadata:
  name: "du-ptp-slave"
spec:
  profile:
  - name: "slave"
    interface: "ens5f1" 2
    ptp4lOpts: "-2 -s --summary_interval -4" 3
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 4
  ptpClockThreshold: 5
    holdOverTimeout: 30 # seconds
    maxOffsetThreshold: 100  # nano seconds
    minOffsetThreshold: -100
```

1

必要に応じて、PtpConfigMaster.yaml、PtpConfigSlave.yaml、または PtpConfigSlaveCvl.yaml のいずれか 1 つを指定できます。PtpConfigSlaveCvl.yaml は、Intel E810 Columbiaville NIC の linuxptp サービスを設定します。group-du-sno-ranGen.yaml および group-du-3node-ranGen.yaml に基づいて設定する場合は、PtpConfigSlave.yaml を使用します。

2

デバイス固有のインターフェイス名。

3

PTP 高速イベントを有効にするには、.spec.sourceFiles.spec.profile の ptp4lOpts に --summary_interval -4 値を追加する必要があります。

4

phc2sysOpts の値が必要です。-m はメッセージを stdout に出力します。linuxptp-daemon DaemonSet はログを解析し、Prometheus メトリックを生成します。

5

オプション: ptpClockThreshold スタンザが存在しない場合は、ptpClockThreshold フィールドにデフォルト値が使用されます。スタンザは、デフォルトの ptpClockThreshold 値を示します。ptpClockThreshold 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。holdOverTimeout は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が FREERUN に変わるまでの時間値 (秒単位) です。maxOffsetThreshold および minOffsetThreshold 設定は、CLOCK_REALTIME (phc2sys) またはマスターオフセット (ptp4l) の値と比較するナノ秒単位のオフセット値を設定します。ptp4l または phc2sys のオフセット値がこの範囲外の場合、PTP クロックの状態が FREERUN に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が LOCKED に設定されます。

3. 以下の PolicyGenTemplate の変更を、特定のサイトの YAML ファイル (例: example-sno-site.yaml) に適用してください。

   a. .sourceFiles に、AMQ ルーターを設定する Interconnect CR ファイルを config-policy に追加します。

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

4. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

5. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- AMQ メッセージングバスのインストール
- OpenShift イメージレジストリーの概要

10.2.9. PolicyGenTemplate CR を使用したベアメタルイベントの設定

GitOps ZTP パイプラインを使用して、HTTP または AMQP トランスポートを使用するベアメタルイベントを設定できます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

10.2.9.1. HTTP トランスポートを使用するベアメタルイベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、HTTP トランスポートを使用するベアメタルイベントを設定できます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. 次の YAML を common-ranGen.yaml ファイルの spec.sourceFiles に追加して、Bare Metal Event Relay Operator を設定します。

```
# Bare Metal Event Relay Operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. たとえば、group-du-sno-ranGen.yaml ファイルの特定のグループ設定ファイルで、HardwareEvent CR を spec.sourceFiles に追加します。

```
- fileName: HardwareEvent.yaml (1)
  policyName: "config-policy"
  spec:
    nodeSelector: {}
    transportHost: "http://hw-event-publisher-service.openshift-bare-metal-
events.svc.cluster.local:9043"
    logLevel: "info"
```

(1) 各ベースボード管理コントローラー (BMC) では、1 つの HardwareEvent CR のみ必要です。

注記

OpenShift Container Platform 4.13 以降では、ベアメタルイベントで HTTP トランスポートを使用する場合、HardwareEvent カスタムリソース (CR) の TransportHost フィールドを設定する必要はありません。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

4. 変更をサイト設定リポジトリーにプッシュし、GitOps ZTP を使用してベアメタルイベントを新しいサイトにデプロイします。

5. 次のコマンドを実行して Redfish シークレットを作成します。

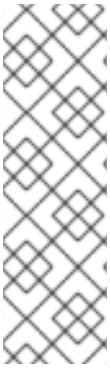
```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

関連情報

- CLI を使用した Bare Metal Event リレーのインストール
- ベアメタルイベントおよびシークレット CR の作成

10.2.9.2. AMQP トランスポートを使用するベアメタルイベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイしたマネージドクラスター上で、AMQP トランスポートを使用するベアメタルイベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。

手順

1. AMQ Interconnect Operator と Bare Metal Event Relay Operator を設定するには、次の YAML を common-ranGen.yaml ファイルの spec.sourceFiles に追加します。

```
# AMQ Interconnect Operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
# Bare Metal Event Relay Operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. Interconnect CR をサイト設定ファイルの .spec.sourceFiles (example-sno-site.yaml ファイルなど) に追加します。

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

3. たとえば、group-du-sno-ranGen.yaml ファイルの特定のグループ設定ファイルで、HardwareEvent CR を spec.sourceFiles に追加します。

```
- path: HardwareEvent.yaml
  patches:
    nodeSelector: {}
    transportHost: "amqp://<amq_interconnect_name>.
<amq_interconnect_namespace>.svc.cluster.local" 1
    logLevel: "info"
```

1

transportHost URL は、既存の AMQ Interconnect CR name と namespace で設定されます。たとえば、transportHost: "amqp://amq-router.amq-router.svc.cluster.local" では、AMQ Interconnect の name と namespace の両方が amq-router に設定されます。

注記

各ベースボード管理コントローラー (BMC) には、単一の HardwareEvent リソースのみが必要です。

4. Git で PolicyGenTemplate の変更をコミットし、その変更をサイト設定リポジトリーにプッシュして、GitOps ZTP を使用してベアメタルイベント監視を新しいサイトにデプロイします。

5. 次のコマンドを実行して Redfish シークレットを作成します。

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password>
\
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

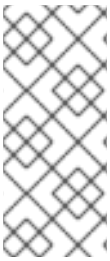
10.2.10. イメージをローカルにキャッシュするための Image Registry Operator の設定

OpenShift Container Platform は、ローカルレジストリーを使用してイメージのキャッシュを管理します。エッジコンピューティングのユースケースでは、クラスターは集中型のイメージレジストリーと通信するときに帯域幅の制限を受けることが多く、イメージのダウンロード時間が長くなる可能性があります。

初期デプロイメント中はダウンロードに時間がかかることは避けられません。時間の経過とともに、予期しないシャットダウンが発生した場合に CRI-O が /var/lib/containers/storage ディレクトリーを消去するリスクがあります。イメージのダウンロード時間が長い場合の対処方法として、GitOps Zero Touch Provisioning (ZTP) を使用してリモートマネージドクラスター上にローカルイメージレジストリーを作成できます。これは、クラスターがネットワークの遠端にデプロイメントされるエッジコンピューティングシナリオで役立ちます。

GitOps ZTP を使用してローカルイメージレジストリーをセットアップする前に、リモートマネージドクラスターのインストールに使用する SiteConfig CR でディスクパーティショニングを設定する必要があります。インストール後、PolicyGenTemplate CR を使用してローカルイメージレジストリーを設

定します。次に、GitOps ZTP パイプラインは永続ボリューム (PV) と永続ボリューム要求 (PVC) CR を作成し、imageregistry 設定にパッチを適用します。



注記

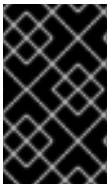
ローカルイメージレジストリーは、ユーザーアプリケーションイメージにのみ使用でき、OpenShift Container Platform または Operator Lifecycle Manager Operator イメージには使用できません。

関連情報

- OpenShift Container Platform レジストリーの概要

10.2.10.1. SiteConfig を使用したディスクパーティショニングの設定

SiteConfig CR と GitOps Zero Touch Provisioning (ZTP) を使用して、マネージドクラスターのディスクパーティションを設定します。SiteConfig CR のディスクパーティションの詳細は、基になるディスクと一致する必要があります。



重要

この手順はインストール時に完了する必要があります。

前提条件

- Butane をインストールします。

手順

1. storage.bu ファイルを作成します。

```
variant: fcos
version: 1.3.0
storage:
  disks:
  - device: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0 (1)
    wipe_table: false
    partitions:
    - label: var-lib-containers
      start_mib: <start_of_partition> (2)
      size_mib: <partition_size> (3)
```

```
filesystems:
  - path: /var/lib/containers
    device: /dev/disk/by-partlabel/var-lib-containers
    format: xfs
    wipe_filesystem: true
    with_mount_unit: true
    mount_options:
      - defaults
      - prjquota
```

1 ルートディスクを指定します。

2 パーティションの開始点を MiB 単位で指定します。値が小さすぎると、インストールに失敗します。

3 パーティションのサイズを指定します。値が小さすぎると、デプロイメントは失敗します。

2. 次のコマンドを実行して、storage.bu を Ignition ファイルに変換します。

```
$ butane storage.bu
```

出力例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# # Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/disk/by-partlabel/var-lib-containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. JSON Pretty Print などのツールを使用して、出力を JSON 形式に変換します。

4. 出力を SiteConfig CR の .spec.clusters.nodes.ignitionConfigOverride フィールドにコピーします。

例

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: |
          {
            "ignition": {
              "version": "3.2.0"
            },
            "storage": {
              "disks": [
                {
                  "device": "/dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0",
                  "partitions": [
                    {
                      "label": "var-lib-containers",
                      "sizeMiB": 0,
                      "startMiB": 250000
                    }
                  ],
                  "wipeTable": false
                }
              ],
              "filesystems": [
                {
                  "device": "/dev/disk/by-partlabel/var-lib-containers",
                  "format": "xfs",
                  "mountOptions": [
                    "defaults",
                    "prjquota"
                  ],
                  "path": "/var/lib/containers",
                  "wipeFilesystem": true
                }
              ]
            },
            "systemd": {
              "units": [
                {
                  "contents": "# # Generated by Butane\n[Unit]\nRequires=systemd-
fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-
fsck@dev-disk-by\\x2dpartlabel-
```

```
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target",
                "enabled": true,
                "name": "var-lib-containers.mount"
              }
            ]
          }
        }
[...]
```

注記

.spec.clusters.nodes.ignitionConfigOverride フィールドが存在しない場合は、作成します。

検証

1. インストール時またはインストール後に、次のコマンドを実行して、BareMetalHost オブジェクトがアノテーションを表示することを確認します。

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-
install.openshift.io/ignition-config-overrides"]
```

出力例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-
id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-
containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":
[{\"device\":\"/dev/disk/by-partlabel/var-lib-
containers\",\"format\":\"xfs\",\"mountOptions\":
[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"sys
temd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-
fsck@dev-disk-by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-
by\\\\x2dpartlabel-
var\\\\x2dlib\\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/
dev/disk/by-partlabel/var-lib-
containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-
fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. インストール後に、単一ノードの OpenShift ディスクのステータスを確認します。

a. 次のコマンドを実行して、シングルノード OpenShift ノードのデバッグセッションに入ります。この手順は、<node_name>-debug というデバッグ Pod をインスタンス化します。

```
$ oc debug node/my-sno-node
```

b. 次のコマンドを実行して、デバッグシェル内のルートディレクトリーとして /host を設定します。デバッグ Pod は、Pod 内の /host にホストの root ファイルシステムをマウントします。root ディレクトリーを /host に変更すると、ホストの実行パスに含まれるバイナリーを実行できます。

```
# chroot /host
```

c. 以下のコマンドを実行して、利用可能なすべてのブロックデバイスに関する情報を一覧表示します。

```
# lsblk
```

出力例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                       /sysroot/ostree/deploy/rhcos/var
│                       /usr
│                       /etc
│                       /
│                       /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

d.

次のコマンドを実行して、ファイルシステムのディスク領域の使用状況に関する情報を表示します。

```
# df -h
```

出力例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

10.2.10.2. PolicyGenTemplate CR を使用してイメージレジストリーを設定する

PolicyGenTemplate (PGT) CR を使用して、イメージレジストリーの設定に必要な CR を適用し、imageregistry 設定にパッチを適用します。

前提条件

- 管理対象クラスターでディスクパーティションを設定しました。
- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- GitOps Zero Touch Provisioning (ZTP) で使用するカスタムサイト設定データを管理する Git リポジトリーを作成している。

手順

1. 適切な PolicyGenTemplate CR で、ストレージクラス、永続ボリューム要求、永続ボ

リューム、およびイメージレジストリー設定を設定します。たとえば、個々のサイトを設定するには、次の YAML をファイル example-sno-site.yaml に追加します。

```
sourceFiles:
  # storage class
  - fileName: StorageClass.yaml
    policyName: "sc-for-image-registry"
    metadata:
      name: image-registry-sc
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100" (1)
  # persistent volume claim
  - fileName: StoragePVC.yaml
    policyName: "pvc-for-image-registry"
    metadata:
      name: image-registry-pvc
      namespace: openshift-image-registry
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
    spec:
      accessModes:
        - ReadWriteMany
      resources:
        requests:
          storage: 100Gi
      storageClassName: image-registry-sc
      volumeMode: Filesystem
  # persistent volume
  - fileName: ImageRegistryPV.yaml (2)
    policyName: "pv-for-image-registry"
    metadata:
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
  - fileName: ImageRegistryConfig.yaml
    policyName: "config-for-image-registry"
    complianceType: musthave
    metadata:
      annotations:
        ran.openshift.io/ztp-deploy-wave: "100"
    spec:
      storage:
        pvc:
          claim: "image-registry-pvc"
```

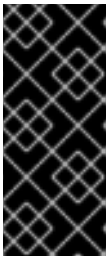
(1)

サイト、共通、またはグループレベルでイメージレジストリーを設定するかどうかに応じて、ztp-deploy-wave に適切な値を設定します。ztp-deploy-wave: "100" は、参照されるソースファイルをグループ化できるため、開発またはテストに適しています。

(2)

ImageRegistryPV.yaml で、spec.local.path フィールドが /var/imageregistry に設定され、SiteConfig CR の mount_point フィールドに設定された値と一致することを確

認します。



重要

- fileName: ImageRegistryConfig.yaml 設定には、complianceType: mustonlyhave を設定しないでください。これにより、レジストリー Pod のデプロイが失敗する可能性があります。

2. Git で PolicyGenTemplate 変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視される Git リポジトリーにプッシュします。

検証

次の手順を使用して、マネージドクラスターのローカルイメージレジストリーに関するエラーをトラブルシューティングします。

- マネージドクラスターにログインしているときに、レジストリーへのログインが成功したことを確認します。以下のコマンドを実行します。

a. マネージドクラスター名をエクスポートします。

```
$ cluster=<managed_cluster_name>
```

b. マネージドクラスター kubeconfig の詳細を取得します。

```
$ oc get secret -n $cluster $cluster-admin-password -o
jsonpath='{.data.password}' | base64 -d > kubeadmin-password-$cluster
```

c. クラスター kubeconfig をダウンロードしてエクスポートします。

```
$ oc get secret -n $cluster $cluster-admin-kubeconfig -o
jsonpath='{.data.kubeconfig}' | base64 -d > kubeconfig-$cluster && export
KUBECONFIG=./kubeconfig-$cluster
```

d. マネージドクラスターからイメージレジストリーへのアクセスを確認します。レジストリーへのアクセスを参照してください。

- imageregistry.operator.openshift.io グループインスタンスの Config CRD がエラーを報告していないことを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get image.config.openshift.io cluster -o yaml
```

出力例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2021-10-08T19:02:39Z"
  generation: 5
  name: cluster
  resourceVersion: "688678648"
  uid: 0406521b-39c0-4cda-ba75-873697da75a4
spec:
  additionalTrustedCA:
    name: acm-ice
```

- 管理対象クラスターの PersistentVolumeClaim にデータが入力されていることを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get pv image-registry-sc
```

- registry* Pod が実行中であり、openshift-image-registry namespace にあることを確認します。

```
$ oc get pods -n openshift-image-registry | grep registry*
```

出力例

```
cluster-image-registry-operator-68f5c9c589-42cfg   1/1     Running     0          8d
image-registry-5f8987879-6nx6h                     1/1     Running     0          8d
```

-

- マネージドクラスターのディスクパーティションが正しいことを確認します。

a. マネージドクラスターへのデバッグシェルを開きます。

```
$ oc debug node/sno-1.example.com
```

b. lsblk を実行して、ホストディスクパーティションを確認します。

```
sh-4.4# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 446.6G  0 disk
  |-sda1   8:1    0     1M  0 part
  |-sda2   8:2    0   127M  0 part
  |-sda3   8:3    0   384M  0 part /boot
  |-sda4   8:4    0 336.3G  0 part /sysroot
  `-sda5   8:5    0 100.1G  0 part /var/imageregistry 1
sdb      8:16   0 446.6G  0 disk
sr0     11:0    1   104M  0 rom
```

1 /var/imageregistry は、ディスクが正しくパーティショニングされていることを示します。

関連情報

- レジストリーへのアクセス

10.3. POLICYGENTEMPLATE リソースと TALM を使用した非接続環境でのマネージドクラスターの更新

Topology Aware Lifecycle Manager (TALM)を使用して、GitOps Zero Touch Provisioning (ZTP) および Topology Aware Lifecycle Manager (TALM)を使用して、デプロイしたマネージドクラスターのソフトウェアライフサイクルを管理できます。TALM は Red Hat Advanced Cluster Management (RHACM) PolicyGenTemplate ポリシーを使用して、ターゲットクラスターに適用される変更を管理および制御します。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および PolicyGenerator CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較
- Topology Aware Lifecycle Manager について

10.3.1. 非接続環境の設定

TALM は、プラットフォームと Operator の更新の両方を実行できます。

TALM を使用して非接続クラスターを更新する前に、ミラーレジストリーで更新するプラットフォームイメージおよび Operator イメージの両方をミラーリングする必要があります。イメージをミラーリングするには以下の手順を実行します。

- プラットフォームの更新では、以下の手順を実行する必要があります。

1. 必要な OpenShift Container Platform イメージリポジトリーをミラーリングします。追加リソースにリンクされている OpenShift Container Platform イメージリポジトリーのミラーリング手順に従って、目的のプラットフォームイメージがミラーリングされていることを確認してください。imageContentSources.yaml ファイルの imageContentSources セクションの内容を保存します。

出力例

```
imageContentSources:
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-release
 - mirrors:
   - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
   source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. ミラーリングされた目的のプラットフォーム イメージのイメージ シグネチャーを保存します。プラットフォームの更新のために、イメージ署名を PolicyGenTemplate CR に追加する必要があります。イメージ署名を取得するには、次の手順を実行します。

   a. 以下のコマンドを実行して、目的の OpenShift Container Platform タグを指定します。

   ```
   $ OCP_RELEASE_NUMBER=<release_version>
   ```

   b. 次のコマンドを実行して、クラスターのアーキテクチャーを指定します。

   ```
   $ ARCHITECTURE=<cluster_architecture> 1
   ```

   1 x86_64、aarch64、s390x、または ppc64le など、クラスターのアーキテクチャーを指定します。

   c. 次のコマンドを実行して、Quay からリリースイメージダイジェストを取得します。

   ```
   $ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From: .*@//p')"
   ```

   d. 次のコマンドを実行して、ダイジェストアルゴリズムを設定します。

   ```
   $ DIGEST_ALGO="${DIGEST%%:*}"
   ```

e. 次のコマンドを実行して、ダイジェスト署名を設定します。

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. 次のコマンドを実行して、mirror.openshift.com Web サイトからイメージ署名を取得します。

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

g. 以下のコマンドを実行して、イメージ署名を checksum-<OCP_RELEASE_NUMBER>.yaml ファイルに保存します。

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 更新グラフを準備します。更新グラフを準備するオプションは 2 つあります。

a. OpenShift Update Service を使用します。

ハブクラスターでグラフを設定する方法の詳細については、 OpenShift Update Service の Operator のデプロイ および グラフデータ init コンテナーのビルド を参照してください。

b. アップストリームグラフのローカルコピーを作成します。マネージドクラスターにアクセスできる非接続環境の http または https サーバーで更新グラフをホストします。更新グラフをダウンロードするには、以下のコマンドを使用します。

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.16 -o ~/upgrade-graph_stable-4.16
```

- Operator の更新については、以下のタスクを実行する必要があります。

  - Operator カタログをミラーリングします。切断されたクラスターで使用する Operator カタログのミラーリングセクションの手順に従って、目的の Operator イメージがミラーリングされていることを確認します。

関連情報

- GitOps ZTP のアップグレード
- OpenShift Container Platform イメージリポジトリーのミラーリング
- 非接続クラスターで使用する Operator カタログのミラーリング
- 非接続環境の準備
- 更新チャネルとリリースについて

10.3.2. PolicyGenTemplate CR を使用したプラットフォーム更新の実行

TALM を使用してプラットフォームの更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- 目的のイメージ リポジトリーをミラーリングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新用の PolicyGenTemplate CR を作成します。

   a. 次の PolicyGenTemplate CR を du-upgrade.yaml ファイルに保存します。

プラットフォーム更新の PolicyGenTemplate の例

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: ImageSignature.yaml 1
      policyName: "platform-upgrade-prep"
      binaryData:
        ${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64} 2
    - fileName: DisconnectedICSP.yaml
      policyName: "platform-upgrade-prep"
      metadata:
        name: disconnected-internal-icsp-for-ocp
      spec:
        repositoryDigestMirrors: 3
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-release
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
    - fileName: ClusterVersion.yaml 4
      policyName: "platform-upgrade"
      metadata:
        name: version
      spec:
        channel: "stable-4.16"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.16
        desiredUpdate:
          version: 4.16.4
      status:
        history:
          - version: 4.16.4
            state: "Completed"
```

1

ConfigMap CR には、更新先の目的のリリースイメージの署名が含まれています。

2

目的の OpenShift Container Platform リリースのイメージ署名を表示します。環境のセットアップセクションの手順に従って保存した checksum-${OCP_RELEASE_NUMBER}.yaml ファイルから署名を取得します。

3

目的の OpenShift Container Platform イメージを含むミラーリポジトリーを表示します。環境のセットアップセクションの手順に従って保存した imageContentSources.yaml ファイルからミラーを取得します。

4

更新をトリガーする ClusterVersion CR を示します。イメージの事前キャッシュには、channel、upstream、および desiredVersion フィールドがすべて必要です。

PolicyGenTemplate CR は 2 つのポリシーを生成します。

- du-upgrade-platform-upgrade-prep ポリシーは、プラットフォームの更新の準備作業を行います。目的のリリースイメージシグネチャーの ConfigMap CR を作成し、ミラー化されたリリースイメージリポジトリーのイメージ コンテンツソースを作成し、目的の更新チャネルと切断された環境でマネージドクラスターが到達可能な更新グラフを使用してクラスターバージョンを更新します。
- du-upgrade-platform-upgrade ポリシーは、プラットフォームのアップグレードを実行するために使用されます。

b. PolicyGenTemplate CR の GitOps ZTP Git リポジトリーにある kustomization.yaml ファイルに du-upgrade.yaml ファイルの内容を追加し、変更を Git リポジトリーにプッシュします。

ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

c. 以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep platform-upgrade
```

2. spec.enable フィールドを false に設定して、プラットフォーム更新用の ClusterGroupUpdate CR を作成します。

a. 次の例に示すように、プラットフォーム更新 ClusterGroupUpdate CR の内容を、du-upgrade-platform-upgrade-prep ポリシーと du-upgrade-platform-upgrade ポリシーおよびターゲットクラスターとともに、cgu-platform-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

b. 次のコマンドを実行して、ClusterGroupUpdate CR をハブクラスターに適用します。

```
$ oc apply -f cgu-platform-upgrade.yml
```

3. オプション: プラットフォームの更新用にイメージを事前キャッシュします。

a. 次のコマンドを実行して、ClusterGroupUpdate CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
platform-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 更新プロセスを監視し、事前キャッシュが完了するまで待ちます。ハブクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

4. プラットフォームの更新を開始します。

a. 次のコマンドを実行して、cgu-platform-upgrade ポリシーを有効にし、事前キャッシュを無効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-
platform-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

関連情報

- 非接続環境の準備

10.3.3. PolicyGenTemplate CR を使用した Operator 更新の実行

TALM で Operator の更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
-

目的のインデックスイメージ、バンドルイメージ、およびバンドルイメージで参照されるすべての Operator イメージをミラーリングします。

- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. Operator の更新用に PolicyGenTemplate CR を更新します。

   a. du-upgrade.yaml ファイルの次の追加コンテンツで du-upgradePolicyGenTemplate CR を更新します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v4.16 1
        updateStrategy: 2
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY 3
```

1. インデックスイメージ URL には、必要な Operator イメージが含まれます。インデックスイメージが常に同じイメージ名とタグにプッシュされている場合、この変更は必要ありません。

2

Operator Lifecycle Manager (OLM) が新しい Operator バージョンのインデックスイメージをポーリングする頻度を registryPoll.interval フィールドで設定します。y-stream および z-stream Operator の更新のために新しいインデックスイメージタグが常にプッシュされる場合、この変更は必要ありません。registryPoll.interval フィールドを短い間隔に設定して更新を促進できますが、間隔を短くすると計算負荷が増加します。これに対処するために、更新が完了したら、registryPoll.interval をデフォルト値に戻すことができます。

3

カタログ接続が最後に監視された状態。READY 値は、CatalogSource ポリシーの準備が整っていることを保証し、インデックス Pod がプルされ、実行中であることを示します。このように、TALM は最新のポリシー準拠状態に基づいて Operator をアップグレードします。

b.

この更新により、1 つのポリシー du-upgrade-operator-catsrc-policy が生成され、必要な Operator イメージを含む新しいインデックスイメージで redhat-operators カタログソースが更新されます。

注記

Operator にイメージの事前キャッシュを使用する必要があり、redhat-operators 以外の別のカタログソースからの Operator がある場合は、次のタスクを実行する必要があります。

- 別のカタログソースの新しいインデックスイメージまたはレジストリーポーリング間隔の更新を使用して、別のカタログソースポリシーを準備します。
- 異なるカタログソースからの目的の Operator に対して個別のサブスクリプションポリシーを準備します。

たとえば、目的の SRIOV-FEC Operator は、certified-operators カタログソースで入手できます。カタログソースと Operator サブスクリプションを更新するには、次の内容を追加して、2 つのポリシー du-upgrade-fec-catsrc-policy と du-upgrade-subscriptions-fec-policy を生成します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
```

```
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    # ...
  - fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "fec-catsrc-policy"
    metadata:
      name: certified-operators
    spec:
      displayName: Intel SRIOV-FEC Operator
      image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
      updateStrategy:
        registryPoll:
          interval: 10m
  - fileName: AcceleratorsSubscription.yaml
    policyName: "subscriptions-fec-policy"
    spec:
      channel: "stable"
      source: certified-operators
```

c.

共通の PolicyGenTemplate CR に指定されたサブスクリプションチャネルが存在する場合は、それらを削除します。GItOps ZTP イメージのデフォルトサブスクリプションチャネルが更新に使用されます。

注記

GItOps ZTP 4.15 で適用される Operator のデフォルトチャネルは、performance-addon-operator を除きすべて stable です。OpenShift Container Platform 4.11 以降、performance-addon-operator 機能は node-tuning-operator に移動されました。4.10 リリースの場合、PAO のデフォルトチャネルは v4.10 です。共通の PolicyGenTemplate CR でデフォルトのチャネルを指定することもできます。

d.

PolicyGenTemplate CR の更新を GitOps ZTP Git リポジトリーにプッシュします。

ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

e.

以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. Operator の更新を開始する前に、必要なカタログソースの更新を適用します。

   a. operator-upgrade-prep という名前の ClusterGroupUpgrade CR の内容をカタログソースポリシーと共に、ターゲットマネージドクラスターの内容を cgu-operator-upgrade-prep.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

   b. 次のコマンドを実行して、ポリシーをハブ クラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

   c. 更新プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. spec.enable フィールドを false に設定して、Operator 更新の ClusterGroupUpgrade CR を作成します。

   a. 以下の例のように、Operator 更新 ClusterGroupUpgrade CR の内容を du-upgrade-operator-catsrc-policy ポリシーで保存して、共通の PolicyGenTemplate およびターゲットクラスターで作成されたサブスクリプションポリシーを cgu-operator-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
```

```
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy 1
  - common-subscriptions-policy 2
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1

このポリシーは、カタログソースから Operator イメージを取得するために、イメージの事前キャッシュ機能で必要になります。

2

ポリシーには Operator サブスクリプションが含まれます。参照 PolicyGenTemplates の構造と内容に従っている場合、すべての Operator サブスクリプションは common-subscriptions-policy ポリシーにグループ化されます。

注記

1 つの ClusterGroupUpgrade CR は、ClusterGroupUpgrade CR に含まれる 1 つのカタログソースからサブスクリプションポリシーで定義される必要な Operator のイメージのみを事前キャッシュできます。SRIOV-FEC Operator の例のように、目的の Operator が異なるカタログソースからのものである場合、別の ClusterGroupUpgrade CR を du-upgrade-fec-catsrc-policy および du-upgrade-subscriptions-fec-policy ポリシーで作成する必要があります。SRIOV-FEC Operator イメージの事前キャッシュと更新。

b. 次のコマンドを実行して、ClusterGroupUpgrade CR をハブクラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. オプション: Operator の更新用にイメージを事前キャッシュします。

a. イメージの事前キャッシュを開始する前に、以下のコマンドを実行して、サブスクリプションポリシーがこの時点で NonCompliant であることを確認します。

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

出力例

```
NAME                          REMEDIATION ACTION   COMPLIANCE STATE     AGE
common-subscriptions-policy   inform               NonCompliant         27d
```

b. 以下のコマンドを実行して、ClusterGroupUpgrade CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

c. プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

d. 以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうかを確認します。

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
```

出力例

```
[
  {
    "lastTransitionTime": "2022-03-08T20:49:08.000Z",
    "message": "The ClusterGroupUpgrade CR is not enabled",
    "reason": "UpgradeNotStarted",
    "status": "False",
    "type": "Ready"
  },
  {
    "lastTransitionTime": "2022-03-08T20:55:30.000Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
```

```
      "type": "PrecachingDone"
    }
]
```

5. Operator の更新を開始します。

a. 以下のコマンドを実行して cgu-operator-upgrade ClusterGroupUpgrade CR を有効にし、事前キャッシュを無効にして Operator の更新を開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

関連情報

- GitOps ZTP のアップグレード

10.3.4. PolicyGenTemplate CR を使用した失敗した Operator の更新のトラブルシューティング

一部のシナリオでは、ポリシーのコンプライアンス状態が古いため、Topology Aware Lifecycle Manager (TALM) が Operator の更新を見逃す可能性があります。

カタログソースの更新後に Operator Lifecycle Manager (OLM) がサブスクリプションステータスを更新すると、時間がかかります。TALM が修復が必要かどうかを判断する間、サブスクリプションポリシーのステータスは準拠していると表示される場合があります。その結果、サブスクリプションポリシーで指定された Operator はアップグレードされません。

このシナリオを回避するには、別のカタログソース設定を PolicyGenTemplate に追加し、更新が必要な Operator のサブスクリプションでこの設定を指定します。

手順

1. PolicyGenTemplate リソースにカタログソース設定を追加します。

```
- fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "operator-catsrc-policy"
    metadata:
      name: redhat-operators
    spec:
      displayName: Red Hat Operators Catalog
      image: registry.example.com:5000/olm/redhat-operators:v{product-version}
      updateStrategy:
        registryPoll:
          interval: 1h
    status:
      connectionState:
        lastObservedState: READY
- fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "operator-catsrc-policy"
    metadata:
      name: redhat-operators-v2 (1)
    spec:
      displayName: Red Hat Operators Catalog v2 (2)
      image: registry.example.com:5000/olredhat-operators:<version> (3)
      updateStrategy:
        registryPoll:
          interval: 1h
    status:
      connectionState:
        lastObservedState: READY
```

(1) 新しい設定の名前を更新します。

(2) 新しい設定の表示名を更新します。

(3) インデックスイメージの URL を更新します。この fileName.spec.image フィールドは、DefaultCatsrc.yaml ファイル内の設定をオーバーライドします。

2. 更新が必要な Operator の新しい設定を指すように Subscription リソースを更新します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: operator-subscription
  namespace: operator-namspace
# ...
spec:
  source: redhat-operators-v2 1
# ...
```

1

PolicyGenTemplate リソースで定義した追加のカタログソース設定の名前を入力します。

10.3.5. プラットフォームと Operator の更新を一緒に実行する

プラットフォームと Operator の更新を同時に実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新の実行および Operator 更新の実行セクションで説明されている手順に従って、更新用の PolicyGenTemplate CR を作成します。
2. プラットフォームの準備作業と Operator の更新を適用します。
   a. プラットフォームの更新の準備作業、カタログ ソースの更新、およびターゲット ク

ラスターのポリシーを 含む ClusterGroupUpgrade CR の内容を cgu-platform-operator-upgrade-prep.yml ファイルに保存します。次に例を示します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

b. 次のコマンドを実行して、cgu-platform-operator-upgrade-prep.yml ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

c. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

3. プラットフォーム用の ClusterGroupUpdate CR と、spec.enable フィールドを false に設定した Operator 更新を作成します。

a. 次の例に示すように、ポリシーとターゲットクラスターを含むプラットフォームと Operator の更新 ClusterGroupUpdate CR の内容を cgu-platform-operator-upgrade.yml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade 1
  - du-upgrade-operator-catsrc-policy 2
  - common-subscriptions-policy 3
```

```
  preCaching: true
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1 これはプラットフォーム更新ポリシーです。

2 これは、更新される Operator のカタログソース情報が含まれるポリシーです。事前キャッシュ機能がマネージドクラスターにダウンロードする Operator イメージを決定するために必要です。

3 これは、Operator を更新するためのポリシーです。

b. 次のコマンドを実行して、cgu-platform-operator-upgrade.yml ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. オプション: プラットフォームおよび Operator の更新用にイメージを事前キャッシュします。

a. 以下のコマンドを実行して、ClusterGroupUpgrade CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 更新プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

c. 以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうか

を確認します。

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5. プラットフォームおよび Operator の更新を開始します。

   a. 以下のコマンドを実行して、cgu-du-upgrade ClusterGroupUpgrade CR がプラットフォームと Operator の更新を開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

   b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

注記

プラットフォームおよび Operator 更新の CR は、設定を spec.enable: true に設定して最初から作成できます。この場合、更新は事前キャッシュが完了した直後に開始し、CR を手動で有効にする必要はありません。

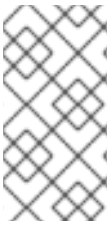
事前キャッシュと更新の両方で、ポリシー、配置バインディング、配置ルール、マネージドクラスターアクション、マネージドクラスタービューなどの追加リソースが作成され、手順を完了することができます。afterCompletion.deleteObjects フィールドを true に設定すると、更新の完了後にこれらのリソースがすべて削除されます。

10.3.6. PolicyGenTemplate CR を使用してデプロイされたクラスターから Performance Addon Operator サブスクリプションを削除する

以前のバージョンの OpenShift Container Platform では、Performance Addon Operator はアプリケーションの自動低レイテンシーパフォーマンスチューニングを提供していました。OpenShift Container Platform 4.11 以降では、これらの機能は Node Tuning Operator の一部です。

OpenShift Container Platform 4.11 以降を実行しているクラスターに Performance Addon Operator をインストールしないでください。OpenShift Container Platform 4.11 以降にアップグレー

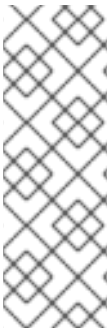
ドすると、Node Tuning Operator は Performance Addon Operator を自動的に削除します。



注記

Operator の再インストールを防ぐために、Performance Addon Operator サブスクリプションを作成するポリシーを削除する必要があります。

参照 DU プロファイルには、PolicyGenTemplate CR common-ranGen.yaml に Performance Addon Operator が含まれています。デプロイされたマネージドクラスターからサブスクリプションを削除するには、common-ranGen.yaml を更新する必要があります。



注記

Performance Addon Operator 4.10.3-5 以降を OpenShift Container Platform 4.11 以降にインストールする場合、Performance Addon Operator はクラスターのバージョンを検出し、Node Tuning Operator 機能との干渉を避けるために自動的に休止状態になります。ただし、最高のパフォーマンスを確保するには、OpenShift Container Platform 4.11 クラスターから Performance Addon Operator を削除してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。
- OpenShift Container Platform 4.11 以降に更新します。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. common-ranGen.yaml ファイル の Performance Addon Operator namespace、Operator グループ、およびサブスクリプションの ComplianceType を mustnothave に変更します。

```
- fileName: PaoSubscriptionNS.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscriptionOperGroup.yaml
```

```
    policyName: "subscriptions-policy"
    complianceType: mustnothave
  - fileName: PaoSubscription.yaml
    policyName: "subscriptions-policy"
    complianceType: mustnothave
```

2. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。common-subscriptions-policy ポリシーのステータスが Non-Compliant に変わります。

3. Topology Aware Lifecycle Manager を使用して、ターゲットクラスターに変更を適用します。設定変更のロールアウトの詳細については、「関連情報」セクションを参照してください。

4. プロセスを監視します。ターゲットクラスターの common-subscriptions-policy ポリシーのステータスが Compliant の場合、Performance Addon Operator はクラスターから削除されています。次のコマンドを実行して、common-subscriptions-policy のステータスを取得します。

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

5. common-ranGen.yaml ファイルの .spec.sourceFiles から Performance Addon Operator namespace、Operator グループ、およびサブスクリプション CR を削除します。

6. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。ポリシーは準拠したままです。

10.3.7. シングルノード OpenShift クラスター上の TALM を使用したユーザー指定のイメージの事前キャッシュ

アプリケーションをアップグレードする前に、アプリケーション固有のワークロードイメージを単一ノード OpenShift クラスターに事前キャッシュできます。

次のカスタムリソース (CR) を使用して、事前キャッシュジョブの設定オプションを指定できます。

- PreCachingConfig CR

- ClusterGroupUpgrade CR

注記

PreCachingConfig CR のフィールドはすべてオプションです。

PreCachingConfig CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: exampleconfig-ns
spec:
  overrides: 1
    platformImage: quay.io/openshift-release-dev/ocp-release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    operatorsIndexes:
      - registry.example.com:5000/custom-redhat-operators:1.0.0
    operatorsPackagesAndChannels:
      - local-storage-operator: stable
      - ptp-operator: stable
      - sriov-network-operator: stable
  spaceRequired: 30 Gi 2
  excludePrecachePatterns: 3
    - aws
    - vsphere
  additionalImages: 4
    - quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    - quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
    - quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
```

1

デフォルトでは、TALM は、マネージドクラスターのポリシーから platformImage、operatorsIndexes、および operatorsPackagesAndChannels フィールドに自動的に値を設定します。これらのフィールドのデフォルトの TALM 派生値をオーバーライドする値を指定できます。

2

クラスター上で最低限必要なディスク容量を指定します。指定しない場合、TALM は OpenShift Container Platform イメージのデフォルト値を定義します。ディスク容量フィールドには、整数値とストレージユニットを含める必要があります。たとえば、40 GiB、200 MB、1 TiB です。

3

イメージ名の一致に基づいて事前キャッシュから除外するイメージを指定します。

4

事前キャッシュする追加イメージのリストを指定します。

PreCachingConfig CR 参照を使用した ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
spec:
  preCaching: true 1
  preCachingConfigRef:
    name: exampleconfig 2
    namespace: exampleconfig-ns 3
```

1

preCaching フィールドを true に設定すると、事前キャッシュジョブが有効になります。

2

preCachingConfigRef.name フィールドは、使用する PreCachingConfig CR を指定します。

3

preCachingConfigRef.namespace は、使用する PreCachingConfig CR の namespace を指定します。

10.3.7.1. 事前キャッシュ用のカスタムリソースの作成

PreCachingConfig CR は、ClusterGroupUpgrade CR の前または同時に作成する必要があります。

1. 事前キャッシュする追加イメージのリストを使用して PreCachingConfig CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: default (1)
spec:
[...]
  spaceRequired: 30Gi (2)
  additionalImages:
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
```

(1) namespace は、ハブクラスターにアクセスできる必要があります。

(2) 事前にキャッシュされたイメージ用に十分なストレージ領域を確保するために、必要な最小ディスク領域フィールドを設定することを推奨します。

2. preCaching フィールドを true に設定して ClusterGroupUpgrade CR を作成し、前の手順で作成した PreCachingConfig CR を指定します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
  namespace: default
spec:
  clusters:
  - sno1
```

```
- sno2
preCaching: true
preCachingConfigRef:
- name: exampleconfig
  namespace: default
managedPolicies:
  - du-upgrade-platform-upgrade
  - du-upgrade-operator-catsrc-policy
  - common-subscriptions-policy
remediationStrategy:
  timeout: 240
```

警告

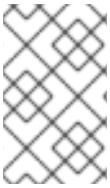
クラスターにイメージをインストールすると、それらを変更したり削除したりすることはできません。

3. イメージを事前キャッシュを開始する場合は、次のコマンドを実行して ClusterGroupUpgrade CR CR を適用します。

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

TALM は ClusterGroupUpgrade CR を検証します。

この時点から、TALM 事前キャッシュワークフローを続行できます。



注記

すべてのサイトが同時に事前キャッシュされます。

検証

1. 次のコマンドを実行して、ClusterUpgradeGroup CR が適用されているハブクラスターの事前キャッシュステータスを確認します。

```
$ oc get cgu <cgu_name> -n <cgu_namespace> -oyaml
```

出力例

```
  precaching:
    spec:
      platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2
e1ef
      operatorsIndexes:
        - registry.example.com:5000/custom-redhat-operators:1.0.0
      operatorsPackagesAndChannels:
        - local-storage-operator: stable
        - ptp-operator: stable
        - sriov-network-operator: stable
      excludePrecachePatterns:
        - aws
        - vsphere
      additionalImages:
        -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47e2e1ef
        -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d297
6d3808fc925ada29c559a47adfaef
        -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340
adfafdedga0991234afdadfsa09
      spaceRequired: "30"
    status:
      sno1: Starting
      sno2: Starting
```

事前キャッシュ設定は、管理ポリシーが存在するかどうかをチェックすることによって検証されます。ClusterGroupUpgrade および PreCachingConfig CR の設定が有効であると、次のステータスになります。

有効な CR の出力例

```
- lastTransitionTime: "2023-01-01T00:00:01Z"
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: "True"
  type: ClusterSelected
- lastTransitionTime: "2023-01-01T00:00:02Z"
  message: Completed validation
```

```
    reason: ValidationCompleted
    status: "True"
    type: Validated
  - lastTransitionTime: "2023-01-01T00:00:03Z"
    message: Precaching spec is valid and consistent
    reason: PrecacheSpecIsWellFormed
    status: "True"
    type: PrecacheSpecValid
  - lastTransitionTime: "2023-01-01T00:00:04Z"
    message: Precaching in progress for 1 clusters
    reason: InProgress
    status: "False"
    type: PrecachingSucceeded
```

無効な PreCachingConfig CR の例

```
Type:    "PrecacheSpecValid"
Status:  False,
Reason:  "PrecacheSpecIncomplete"
Message: "Precaching spec is incomplete: failed to get PreCachingConfig resource
due to PreCachingConfig.ran.openshift.io "<pre-caching_cr_name>" not found"
```

2. マネージドクラスターで次のコマンドを実行すると、事前キャッシュジョブを見つけることができます。

```
$ oc get jobs -n openshift-talo-pre-cache
```

進行中の事前キャッシュジョブの例

```
NAME        COMPLETIONS      DURATION      AGE
pre-cache   0/1              1s            1s
```

3. 次のコマンドを実行して、事前キャッシュジョブ用に作成された Pod のステータスを確認できます。

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

進行中の事前キャッシュジョブの例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate     19s    job-controller    Created pod: pre-cache-abcd1
```

4. 次のコマンドを実行すると、ジョブのステータスに関するライブ更新を取得できます。

```
$ oc logs -f pre-cache-abcd1 -n openshift-talo-pre-cache
```

5. 事前キャッシュジョブが正常に完了したことを確認するには、次のコマンドを実行します。

```
$ oc describe pod pre-cache -n openshift-talo-pre-cache
```

完了した事前キャッシュジョブの例

```
Type       Reason            Age    From            Message
Normal     SuccesfulCreate     5m19s  job-controller    Created pod: pre-cache-abcd1
Normal     Completed           19s    job-controller    Job completed
```

6. イメージが単一ノード OpenShift で正常に事前キャッシュ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. デバッグモードでノードに入ります。

   ```
   $ oc debug node/cnfdf00.example.lab
   ```

   b. root を host に変更します。

   -

```
$ chroot /host/
```

c. 目的のイメージを検索します。

```
$ sudo podman images | grep <operator_name>
```

関連情報

- コンテナーイメージ事前キャッシュ機能の使用

10.3.8. GitOps ZTP 用に自動作成された ClusterGroupUpgrade CR について

TALM には、ManagedClusterForCGU と呼ばれるコントローラーがあります。このコントローラーは、ハブクラスター上で ManagedCluster CR の Ready 状態を監視し、GitOps Zero Touch Provisioning (ZTP) の ClusterGroupUpgrade CR を作成します。

ztp-done ラベルが適用されていない Ready 状態のマネージドクラスターの場合、ManagedClusterForCGU コントローラーは、ztp-install namespace に ClusterGroupUpgrade CR と、GItOps ZTP プロセス中に作成された関連する RHACM ポリシーを自動的に作成します。次に TALM は自動作成された ClusterGroupUpgrade CR に一覧表示されている設定ポリシーのセットを修正し、設定 CR をマネージドクラスターにプッシュします。

クラスターが Ready になった時点でマネージドクラスターのポリシーがない場合、ポリシーのない ClusterGroupUpgrade CR が作成されます。ClusterGroupUpgrade が完了すると、マネージドクラスターには ztp-done というラベルが付けられます。そのマネージドクラスターに適用するポリシーがある場合は、2 日目の操作として ClusterGroupUpgrade を 手動で作成します。

GitOps ZTP 用に自動作成された ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
```

```
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" (1)
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" (2)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

(1)

TALM がクラスター設定を完了する際にマネージドクラスターに適用されます。

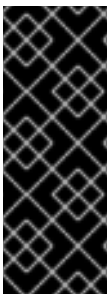
(2)

TALM が設定ポリシーのデプロイを開始するときにマネージドクラスターに適用されます。

第11章 POLICYGENERATOR または POLICYGENTEMPLATE CR でのハブテンプレートの使用

Topology Aware Lifecycle Manager は、GitOps Zero Touch Provisioning (ZTP) で使用される設定ポリシーで、部分的な Red Hat Advanced Cluster Management (RHACM) ハブクラスターテンプレート機能をサポートします。

ハブ側のクラスターテンプレートを使用すると、ターゲットクラスターに合わせて動的にカスタマイズできる設定ポリシーを定義できます。これにより、設定は似ているが値が異なる多くのクラスターに対して個別のポリシーを作成する必要がなくなります。



重要

ポリシーテンプレートは、ポリシーが定義されている namespace と同じ namespace に制限されています。これは、ハブテンプレートで参照されるオブジェクトを、ポリシーが作成されたのと同じ namespace に作成する必要があることを意味します。

重要

PolicyGenTemplate CR を使用してマネージドクラスターへのポリシーを管理およびデプロイすることは、今後の OpenShift Container Platform リリースで非推奨になりました。同等の機能および改善された機能は、Red Hat Advanced Cluster Management (RHACM)および PolicyGenerator CR を使用して利用できます。

PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

関連情報

- PolicyGenerator リソースを使用したマネージドクラスターポリシーの設定
- RHACM PolicyGenerator と PolicyGenTemplate リソースパッチの比較

11.1. 設定ポリシーでの RHACM ハブクラスターテンプレートの使用

Topology Aware Lifecycle Manager は、GitOps Zero Touch Provisioning (ZTP) で使用される設定ポリシーで、部分的な Red Hat Advanced Cluster Management (RHACM) ハブクラスターテンプレート機能をサポートします。

TALM を使用する GitOps ZTP では、次のサポートされているハブテンプレート関数を使用できます。

- fromConfigmap は、指定された ConfigMap リソースで提供されたデータキーの値を返します。

注記

ConfigMap CR には 1 MiB のサイズ制限 があります。ConfigMap CR の有効サイズは、last-applied-configuration アノテーションによってさらに制限されます。last-applied-configuration 制限を回避するには、次のアノテーションをテンプレート ConfigMap に追加します。

```
argocd.argoproj.io/sync-options: Replace=true
```

- base64enc は、base64 でエンコードされた入力文字列の値を返します
- base64dec は、base64 でエンコードされた入力文字列のデコードされた値を返します
- indent は、インデントスペースが追加された入力文字列を返します
- autoindent は、親テンプレートで使用されているスペースに基づいてインデントスペースを追加した入力文字列を返します。
- toInt は、入力値の整数値をキャストして返します
- toBool は入力文字列をブール値に変換し、ブール値を返します

GitOps ZTP では、さまざまな オープンソースコミュニティー機能 も利用できます。

関連情報

- 設定ポリシーでのハブクラスターテンプレートの RHACM サポート

11.2. ハブテンプレートの例

次のコード例は、有効なハブテンプレートです。これらの各テンプレートは、default namespace で test-config という名前の ConfigMap CR から値を返します。

- キー common-key を持つ値を返します。

```
{{hub fromConfigMap "default" "test-config" "common-key" hub}}
```

- .ManagedClusterName フィールドと文字列 -name の連結値を使用して、文字列を返します。

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) hub}}
```

- .ManagedClusterName フィールドと文字列 -name の連結値からブール値をキャストして返します。

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) | toBool hub}}
```

- .ManagedClusterName フィールドと文字列 -name の連結値から整数値をキャストして返します。

```
{{hub (printf "%s-name" .ManagedClusterName) | fromConfigMap "default" "test-config" | toInt hub}}
```

11.3. グループ POLICYGENERATOR または POLICYGENTEMPLATE CR でのグループおよびサイト設定の指定

ハブテンプレートを使用して、マネージドクラスターに適用される生成済みポリシーにグループとサイトの値を入力することで、ConfigMap CR でクラスターのフリート設定を管理できます。サイト PolicyGenerator または PolicyGentemplate CR でハブテンプレートを使用すると、サイトごとにポリシー CR を作成する必要がなくなります。

ハードウェアの種類や地域などのユースケースに応じて、フリート内のクラスターをさまざまなカテゴリーにグループ化できます。各クラスターには、そのクラスターが属するグループ (複数可) に対応するラベルが必要です。各グループの設定値を異なる ConfigMap CR で管理する場合、ハブテンプレー

トを使用してグループ内のすべてのクラスターに変更を適用するには、1 つのグループ PolicyGenTemplate CR のみ必要です。

次の例は、3 つの ConfigMap CR と 1 つの PolicyGenTemplate CR グループを使用して、サイトとグループの両方の設定を、ハードウェアタイプとリージョンごとにグループ化されたクラスターに適用する方法を示しています。

注記

fromConfigmap 関数を使用する場合、printf 変数はテンプレートリソース data キーフィールドでのみ使用できます。name および namespace フィールドでは使用できません。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセスでき、GitOps ZTP ArgoCD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. グループとサイトの設定を含む 3 つの ConfigMap CR を作成します。
   a. group-hardware-types-configmap という名前の ConfigMap CR を作成して、ハードウェア固有の設定を保持します。以下に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: group-hardware-types-configmap
  namespace: ztp-group
  annotations:
    argocd.argoproj.io/sync-options: Replace=true (1)
data:
  # SriovNetworkNodePolicy.yaml
  hardware-type-1-sriov-node-policy-pfNames-1: "[\"ens5f0\"]"
```

```
  hardware-type-1-sriov-node-policy-pfNames-2: "[\"ens7f0\"]"
  # PerformanceProfile.yaml
  hardware-type-1-cpu-isolated: "2-31,34-63"
  hardware-type-1-cpu-reserved: "0-1,32-33"
  hardware-type-1-hugepages-default: "1G"
  hardware-type-1-hugepages-size: "1G"
  hardware-type-1-hugepages-count: "32"
```

1

argocd.argoproj.io/sync-options アノテーションは、ConfigMap のサイズが 1 MiB より大きい場合にのみ必要です。

b. group-zones-configmap という名前の ConfigMap CR を作成して、地域設定を保持します。以下に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: group-zones-configmap
  namespace: ztp-group
data:
  # ClusterLogForwarder.yaml
  zone-1-cluster-log-fwd-outputs: "[{\"type\":\"kafka\", \"name\":\"kafka-open\",
\"url\":\"tcp://10.46.55.190:9092/test\"}]"
  zone-1-cluster-log-fwd-pipelines: "[{\"inputRefs\":[\"audit\", \"infrastructure\"],
\"labels\": {\"label1\": \"test1\", \"label2\": \"test2\", \"label3\": \"test3\", \"label4\":
\"test4\"}, \"name\": \"all-to-default\", \"outputRefs\": [\"kafka-open\"]}]"
```

c. site-data-configmap という名前の ConfigMap CR を作成して、サイト固有の設定を保持します。以下に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: site-data-configmap
  namespace: ztp-group
data:
  # SriovNetwork.yaml
  du-sno-1-zone-1-sriov-network-vlan-1: "140"
  du-sno-1-zone-1-sriov-network-vlan-2: "150"
```

注記

各 ConfigMap CR は、PolicyGenTemplate CR グループから生成されるポリシーと同じ namespace に配置される必要があります。

2. Git で ConfigMap CR をコミットし、Argo CD アプリケーションが監視する Git リポジトリーにプッシュします。

3. ハードウェアタイプとリージョンラベルをクラスターに適用します。次のコマンドは、du-sno-1-zone-1 という名前のシングルクラスターに適用され、"hardware-type": "hardware-type-1" および "group-du-sno-zone": "zone-1" のラベルが選択されます。

```
$ oc patch managedclusters.cluster.open-cluster-management.io/du-sno-1-zone-1 --type merge -p '{"metadata":{"labels":{"hardware-type": "hardware-type-1", "group-du-sno-zone": "zone-1"}}}'
```

4. 要件に応じて、ハブテンプレートを使用して ConfigMap オブジェクトから必要なデータを取得するグループ PolicyGenerator または PolicyGentemplate CR を作成します。

   a. グループ PolicyGenerator CR を作成します。この PolicyGenerator CR の例では、policyDefaults.placement フィールドに一覧表示されるラベルと一致するクラスターのロギング、VLAN ID、NIC、および Performance Profile を設定します。

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno-pgt
placementBindingDefaults:
  name: group-du-sno-pgt-placement-binding
policyDefaults:
  placement:
    labelSelector:
      matchExpressions:
        - key: group-du-sno-zone
          operator: In
          values:
            - zone-1
        - key: hardware-type
          operator: In
          values:
            - hardware-type-1
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
    compliant: 10m
    noncompliant: 10s
policies:
```

```
    - name: group-du-sno-pgt-group-du-sno-cfg-policy
      policyAnnotations:
        ran.openshift.io/ztp-deploy-wave: "10"
      manifests:
        - path: source-crs/ClusterLogForwarder.yaml
          patches:
            - spec:
                outputs: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-outputs" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
                pipelines: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-pipelines" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
        - path: source-crs/PerformanceProfile-MCP-master.yaml
          patches:
            - metadata:
                name: openshift-node-performance-profile
              spec:
                additionalKernelArgs:
                  - rcupdate.rcu_normal_after_boot=0
                  - vfio_pci.enable_sriov=1
                  - vfio_pci.disable_idle_d3=1
                  - efi=runtime
                cpu:
                  isolated: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-isolated" (index .ManagedClusterLabels "hardware-type")) hub}}'
                  reserved: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-reserved" (index .ManagedClusterLabels "hardware-type")) hub}}'
                hugepages:
                  defaultHugepagesSize: '{{hub fromConfigMap "" "group-hardware-
types-configmap" (printf "%s-hugepages-default" (index .ManagedClusterLabels
"hardware-type")) hub}}'
                  pages:
                    - count: '{{hub fromConfigMap "" "group-hardware-types-
configmap" (printf "%s-hugepages-count" (index .ManagedClusterLabels
"hardware-type")) | toInt hub}}'
                      size: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-hugepages-size" (index .ManagedClusterLabels "hardware-type"))
hub}}'
                realTimeKernel:
                  enabled: true
    - name: group-du-sno-pgt-group-du-sno-sriov-policy
      policyAnnotations:
        ran.openshift.io/ztp-deploy-wave: "100"
      manifests:
        - path: source-crs/SriovNetwork.yaml
          patches:
            - metadata:
                name: sriov-nw-du-fh
              spec:
                resourceName: du_fh
                vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-1" .ManagedClusterName) | toInt hub}}'
        - path: source-crs/SriovNetworkNodePolicy-MCP-master.yaml
          patches:
            - metadata:
```

```
            name: sriov-nnp-du-fh
          spec:
            deviceType: netdevice
            isRdma: false
            nicSelector:
              pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-1" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
            numVfs: 8
            priority: 10
            resourceName: du_fh
      - path: source-crs/SriovNetwork.yaml
        patches:
        - metadata:
            name: sriov-nw-du-mh
          spec:
            resourceName: du_mh
            vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-2" .ManagedClusterName) | toInt hub}}'
      - path: source-crs/SriovNetworkNodePolicy-MCP-master.yaml
        patches:
        - metadata:
            name: sriov-nw-du-fh
          spec:
            deviceType: netdevice
            isRdma: false
            nicSelector:
              pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-2" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
            numVfs: 8
            priority: 10
            resourceName: du_fh
```

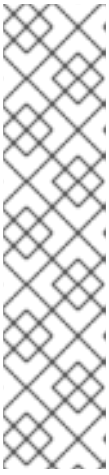
b.

グループ PolicyGenTemplate CR を作成します。例として挙げたこの PolicyGenTemplate CR は、spec.bindingRules の下にリストされているラベルに一致するクラスターのロギング、VLAN ID、NIC、およびパフォーマンスプロファイルを設定します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: group-du-sno-pgt
  namespace: ztp-group
spec:
  bindingRules:
    # These policies will correspond to all clusters with these labels
    group-du-sno-zone: "zone-1"
    hardware-type: "hardware-type-1"
  mcp: "master"
  sourceFiles:
    - fileName: ClusterLogForwarder.yaml # wave 10
      policyName: "group-du-sno-cfg-policy"
      spec:
```

```
      outputs: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-outputs" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'
      pipelines: '{{hub fromConfigMap "" "group-zones-configmap" (printf "%s-
cluster-log-fwd-pipelines" (index .ManagedClusterLabels "group-du-sno-zone")) |
toLiteral hub}}'

  - fileName: PerformanceProfile.yaml # wave 10
    policyName: "group-du-sno-cfg-policy"
    metadata:
      name: openshift-node-performance-profile
    spec:
      additionalKernelArgs:
      - rcupdate.rcu_normal_after_boot=0
      - vfio_pci.enable_sriov=1
      - vfio_pci.disable_idle_d3=1
      - efi=runtime
      cpu:
        isolated: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-cpu-isolated" (index .ManagedClusterLabels "hardware-type")) hub}}'
        reserved: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-cpu-reserved" (index .ManagedClusterLabels "hardware-type")) hub}}'
      hugepages:
        defaultHugepagesSize: '{{hub fromConfigMap "" "group-hardware-types-
configmap" (printf "%s-hugepages-default" (index .ManagedClusterLabels
"hardware-type")) hub}}'
        pages:
          - size: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-hugepages-size" (index .ManagedClusterLabels "hardware-type")) hub}}'
            count: '{{hub fromConfigMap "" "group-hardware-types-configmap" (printf
"%s-hugepages-count" (index .ManagedClusterLabels "hardware-type")) | toInt
hub}}'
      realTimeKernel:
        enabled: true

  - fileName: SriovNetwork.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-fh
    spec:
      resourceName: du_fh
      vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-1" .ManagedClusterName) | toInt hub}}'

  - fileName: SriovNetworkNodePolicy.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nnp-du-fh
    spec:
      deviceType: netdevice
      isRdma: false
      nicSelector:
        pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-1" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
      numVfs: 8
```

```
      priority: 10
      resourceName: du_fh

  - fileName: SriovNetwork.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-mh
    spec:
      resourceName: du_mh
      vlan: '{{hub fromConfigMap "" "site-data-configmap" (printf "%s-sriov-
network-vlan-2" .ManagedClusterName) | toInt hub}}'

  - fileName: SriovNetworkNodePolicy.yaml # wave 100
    policyName: "group-du-sno-sriov-policy"
    metadata:
      name: sriov-nw-du-fh
    spec:
      deviceType: netdevice
      isRdma: false
      nicSelector:
        pfNames: '{{hub fromConfigMap "" "group-hardware-types-configmap"
(printf "%s-sriov-node-policy-pfNames-2" (index .ManagedClusterLabels
"hardware-type")) | toLiteral hub}}'
      numVfs: 8
      priority: 10
      resourceName: du_fh
```



注記

サイト固有の設定値を取得するには、.ManagedClusterName フィールドを使用します。これは、ターゲットマネージドクラスターの名前に設定されたテンプレートコンテキスト値です。

グループ固有の設定を取得するには、.ManagedClusterLabels フィールドを使用します。これは、マネージドクラスターのラベルの値に設定されたテンプレートコンテキスト値です。

5. サイトの PolicyGenTemplate CR を Git にコミットし、ArgoCD アプリケーションによって監視されている Git リポジトリーにプッシュします。

注記

参照された ConfigMap CR に対するその後の変更は、適用されたポリシーに自動的に同期されません。新しい ConfigMap の変更を手動で同期して、既存の PolicyGenTemplate CR を更新する必要があります。「新しい ConfigMap の変更を既存の PolicyGenerator または PolicyGenTemplate CR に同期する」を参照してください。

複数のクラスターに同じ PolicyGenerator または PolicyGentemplate CR を使用できます。設定に変更がある場合、各クラスターの設定とマネージドクラスターのラベルを保持する ConfigMap オブジェクトのみ変更する必要があります。

11.4. 新規 CONFIGMAP の変更を既存の POLICYGENERATOR または POLICYGENTEMPLATE CR に同期する

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてハブクラスターにログインしている。
- ハブクラスターテンプレートを使用して ConfigMap CR から情報を取得する PolicyGenerator または PolicyGentemplate CR を作成している。

手順

1. ConfigMap CR の内容を更新し、変更をハブクラスターに適用します。
2. 更新された ConfigMap CR の内容をデプロイされたポリシーに同期するには、次のいずれかを実行します。
   a. オプション 1: 既存のポリシーを削除します。ArgoCD は PolicyGenerator または PolicyGentemplate CR を使用して、削除されたポリシーをすぐに再作成します。たとえば、以下のコマンドを実行します。

```
$ oc delete policy <policy_name> -n <policy_namespace>
```

b.

オプション 2: ConfigMap を更新するたびに、特別なアノテーション policy.open-cluster-management.io/trigger-update を異なる値でポリシーに適用します。以下に例を示します。

```
$ oc annotate policy <policy_name> -n <policy_namespace> policy.open-cluster-management.io/trigger-update="1"
```

注記

変更を有効にするには、更新されたポリシーを適用する必要があります。詳細については、再処理のための特別なアノテーション を参照してください。

3.

オプション: 存在する場合は、ポリシーを含む ClusterGroupUpdate CR を削除します。以下に例を示します。

```
$ oc delete clustergroupupgrade <cgu_name> -n <cgu_namespace>
```

a.

更新された ConfigMap の変更を適用するポリシーを含む新しい ClusterGroupUpdate CR を作成します。たとえば、次の YAML をファイル cgr-example.yaml に追加します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: <cgr_name>
  namespace: <policy_namespace>
spec:
  managedPolicies:
    - <managed_policy>
  enable: true
  clusters:
  - <managed_cluster_1>
  - <managed_cluster_2>
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

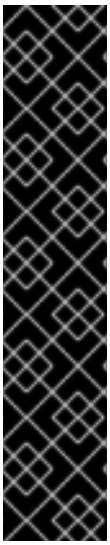
b.

更新されたポリシーを適用します。

```
$ oc apply -f cgr-example.yaml
```

第12章 TOPOLOGY AWARE LIFECYCLE MANAGER を使用したマネージドクラスターの更新

Topology Aware Lifecycle Manager (TALM) を使用して、複数のクラスターのソフトウェアライフサイクルを管理できます。TALM は Red Hat Advanced Cluster Management (RHACM) ポリシーを使用して、ターゲットクラスター上で変更を実行します。



重要

GitOps ZTP での PolicyGenerator リソースの使用は、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 製品サポート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場合があります。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。

12.1. TOPOLOGY AWARE LIFECYCLE MANAGER の設定について

Topology Aware Lifecycle Manager (TALM) は、1 つまたは複数の OpenShift Container Platform クラスターに対する Red Hat Advanced Cluster Management (RHACM) ポリシーのデプロイメントを管理します。TALM を大規模なクラスターのネットワークで使用することにより、限られたバッチで段階的にポリシーをクラスターにデプロイメントすることができます。これにより、更新時のサービス中断の可能性を最小限に抑えることができます。TALM では、以下の動作を制御することができます。

- 更新のタイミング
- RHACM マネージドクラスター数
- ポリシーを適用するマネージドクラスターのサブセット
- クラスターの更新順序
- クラスターに修正されたポリシーのセット

- クラスターに修正されるポリシーの順序
- カナリアクラスターの割り当て

シングルノードの OpenShift の場合、Topology Aware Lifecycle Manager (TALM) は次の機能を提供します。

- アップグレード前に、デプロイメントのバックアップを作成する
- 帯域幅が制限されたクラスターのイメージの事前キャッシュ

TALM は、OpenShift Container Platform y-stream および z-stream 更新のオーケストレーションをサポートし、y-streams および z-streams での day-two 操作をサポートします。

12.2. TOPOLOGY AWARE LIFECYCLE MANAGER で使用される管理ポリシー

Topology Aware Lifecycle Manager (TALM) は、クラスターの更新に RHACM ポリシーを使用します。

TALM は、remediationAction フィールドが inform に設定されているポリシー CR のロールアウトを管理するために使用できます。サポートされるユースケースには、以下が含まれます。

- ポリシー CR の手動ユーザー作成
- PolicyGenerator または PolicyGentemplate カスタムリソース定義(CRD)から自動生成されたポリシー

手動承認で Operator 契約を更新するポリシーのために、TALM は、更新された Operator のインストールを承認する追加機能を提供します。

マネージドポリシーの詳細は、RHACM のドキュメントの ポリシーの概要 を参照してください。

関連情報

関連情報

- PolicyGenerator CRD について

12.3. WEB コンソールを使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール

OpenShift Container Platform Web コンソールを使用して Topology Aware Lifecycle Manager をインストールできます。

前提条件

- 最新バージョンの RHACM Operator をインストールします。
- TALM 4.16 には RHACM 2.9 以降が必要です。
- 非接続の regitry でハブクラスターを設定します。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Container Platform Web コンソールで、Operators → OperatorHub ページに移動します。
2. 利用可能な Operator のリストから Topology Aware Lifecycle Manager を検索し、Install をクリックします。
3. Installation mode ["All namespaces on the cluster (default)"] および Installed Namespace ("openshift-operators") のデフォルトの選択を維持し、Operator が適切にインストールされていることを確認します。
4. Install をクリックします。

検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. Operators → Installed Operators ページに移動します。

2. Operator が All Namespaces ネームスペースにインストールされ、そのステータスが Succeeded であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. Operators → Installed Operators ページに移動し、Status 列でエラーまたは失敗の有無を確認します。

2. Workloads → Pods ページに移動し、問題を報告している cluster-group-upgrades-controller-manager Pod のコンテナーのログを確認します。

12.4. CLI を使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール

OpenShift CLI (oc) を使用して Topology Aware Lifecycle Manager (TALM) をインストールできます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- 最新バージョンの RHACM Operator をインストールします。
- TALM 4.16 には RHACM 2.9 以降が必要です。
- 非接続の regitry でハブクラスターを設定します。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. **SubscriptionCR を作成します。**

   a. **Subscription CR を定義し、YAML ファイルを保存します (例: talm-subscription.yaml)。**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-topology-aware-lifecycle-manager-subscription
  namespace: openshift-operators
spec:
  channel: "stable"
  name: topology-aware-lifecycle-manager
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

   b. **以下のコマンドを実行して Subscription CR を作成します。**

```
$ oc create -f talm-subscription.yaml
```

検証

1. **CSV リソースを調べて、インストールが成功したことを確認します。**

```
$ oc get csv -n openshift-operators
```

出力例

```
NAME                                         DISPLAY                       VERSION
REPLACES                       PHASE
topology-aware-lifecycle-manager.4.16.x   Topology Aware Lifecycle Manager   4.16.x
Succeeded
```

2. **TALM が稼働していることを確認します。**

```
$ oc get deploy -n openshift-operators
```

出力例

```
NAMESPACE                              NAME                                        READY   UP-TO-
DATE   AVAILABLE   AGE
openshift-operators                    cluster-group-upgrades-controller-manager
1/1    1           1         14s
```

12.5. CLUSTERGROUPUPGRADE CR

Topology Aware Lifecycle Manager (TALM) は、クラスター グループの ClusterGroupUpgrade CR から修復計画を作成します。ClusterGroupUpgrade CR で次の仕様を定義できます。

- グループのクラスター
- ClusterGroupUpgrade CR のブロック
- 管理ポリシーの適用リスト
- 同時更新の数
- 適用可能なカナリア更新
- 更新前後に実行するアクション
- 更新タイミング

ClusterGroupUpgrade CR の enable フィールドを使用して、更新の開始時刻を制御できます。たとえば、メンテナンスウィンドウが 4 時間にスケジュールされている場合、enable フィールドを false に設定して ClusterGroupUpgrade CR を準備できます。

次のように spec.remediationStrategy.timeout 設定を設定することで、タイムアウトを設定できます。

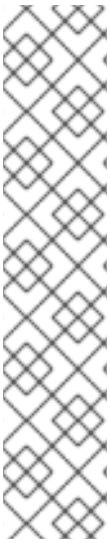
```
spec
  remediationStrategy:
        maxConcurrency: 1
        timeout: 240
```

batchTimeoutAction を使用して、クラスターの更新が失敗した場合にどうなるかを判断できます。continue を指定して失敗したクラスターをスキップし、他のクラスターのアップグレードを続行するか、abort を指定してすべてのクラスターのポリシー修正を停止することができます。タイムアウトが経過すると、TALM はすべての enforce ポリシーを削除して、クラスターがそれ以上更新されないようにします。

変更を適用するには、enabled フィールドを true に設定します。

詳細については、管理対象クラスターへの更新ポリシーの適用セクションを参照してください。

TALM は指定されたクラスターへのポリシーの修復を通じて機能するため、ClusterGroupUpgrade CR は多くの条件について true または false のステータスを報告できます。



注記

TALM がクラスターの更新を完了した後、同じ ClusterGroupUpgrade CR の制御下でクラスターが再度更新されることはありません。次の場合は、新しい ClusterGroupUpgrade CR を作成する必要があります。

- クラスターを再度更新する必要がある場合
- クラスターが更新後に inform ポリシーで非準拠に変更された場合

12.5.1. クラスターの選択

TALM は修復計画を作成し、次のフィールドに基づいてクラスターを選択します。

- clusterLabelSelector フィールドは、更新するクラスターのラベルを指定します。これは、k8s.io/apimachinery/pkg/apis/meta/v1 からの標準ラベルセレクターのリストで設定され

ます。リスト内の各セレクターは、ラベル値ペアまたはラベル式のいずれかを使用します。各セレクターからの一致は、clusterSelector フィールドおよび cluster フィールドからの一致と共に、クラスターの最終リストに追加されます。

- clusters フィールドは、更新するクラスターのリストを指定します。
- canaries フィールドは、カナリア更新のクラスターを指定します。
- maxConcurrency フィールドは、バッチで更新するクラスターの数を指定します。
- actions フィールドは、更新プロセスを開始するときに TALM が実行する beforeEnable アクションと、各クラスターのポリシー修復を完了するときに TALM が実行する afterCompletion アクションを指定します。

clusters、clusterLabelSelector、および clusterSelector フィールドを一緒に使用して、クラスターの結合リストを作成できます。

修復計画は、canaries フィールドにリストされているクラスターから開始されます。各カナリアクラスターは、単一クラスターバッチを形成します。

有効な field が false に設定されたサンプル ClusterGroupUpgrade CR

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion: 1
      addClusterLabels:
        upgrade-done: ""
      deleteClusterLabels:
        upgrade-running: ""
```

```
      deleteObjects: true
    beforeEnable: 2
      addClusterLabels:
        upgrade-running: ""
  backup: false
  clusters: 3
    - spoke1
  enable: false 4
  managedPolicies: 5
    - talm-policy
  preCaching: false
  remediationStrategy: 6
    canaries: 7
      - spoke1
    maxConcurrency: 2 8
    timeout: 240
  clusterLabelSelectors: 9
    - matchExpressions:
      - key: label1
      operator: In
      values:
        - value1a
        - value1b
  batchTimeoutAction: 10
status: 11
    computedMaxConcurrency: 2
    conditions:
      - lastTransitionTime: '2022-11-18T16:27:15Z'
        message: All selected clusters are valid
        reason: ClusterSelectionCompleted
        status: 'True'
        type: ClustersSelected 12
      - lastTransitionTime: '2022-11-18T16:27:15Z'
        message: Completed validation
        reason: ValidationCompleted
        status: 'True'
        type: Validated 13
      - lastTransitionTime: '2022-11-18T16:37:16Z'
        message: Not enabled
        reason: NotEnabled
        status: 'False'
        type: Progressing
    managedPoliciesForUpgrade:
      - name: talm-policy
        namespace: talm-namespace
    managedPoliciesNs:
      talm-policy: talm-namespace
    remediationPlan:
      - - spoke1
      - - spoke2
        - spoke3
    status:
```

1

各クラスターのポリシー修正が完了したときに TALM が実行するアクションを指定します。

2

更新プロセスを開始するときに TALM が実行するアクションを指定します。

3

更新するクラスターの一覧を定義します。

4

enable フィールドは false に設定されています。

5

修正するユーザー定義のポリシーセットを一覧表示します。

6

クラスター更新の詳細を定義します。

7

カナリア更新のクラスターを定義します。

8

バッチの同時更新の最大数を定義します。修復バッチの数は、カナリアクラスターの数に加えて、カナリアクラスターを除くクラスターの数を maxConcurrency 値で除算します。すべての管理ポリシーに準拠しているクラスターは、修復計画から除外されます。

9

クラスターを選択するためのパラメーターを表示します。

10

バッチがタイムアウトした場合の動作を制御します。可能な値は abort または continue です。指定しない場合、デフォルトは continue です。

11

更新のステータスに関する情報を表示します。

12

ClustersSelected 条件は、選択されたすべてのクラスターが有効であることを示します。

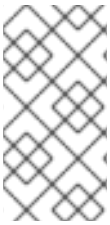
13

Validated 条件は、選択したすべてのクラスターが検証済みであることを示します。

注記

カナリアクラスターの更新中に障害が発生すると、更新プロセスが停止します。

修復計画が正常に作成されたら、enable フィールドを true に設定できます。TALM は、指定された管理ポリシーを使用して、準拠していないクラスターの更新を開始します。



注記

ClusterGroupUpgrade CR の enable フィールドが false に設定されている場合にのみ、spec フィールドを変更できます。

12.5.2. Validating

TALM は、指定されたすべての管理ポリシーが使用可能で正しいことを確認し、Validated 条件を使用して、ステータスと理由を次のようにレポートします。

- true

  検証が完了しました。

- false

  ポリシーが見つからないか無効であるか、無効なプラットフォームイメージが指定されています。

12.5.3. 事前キャッシュ

クラスターにはコンテナーイメージレジストリーにアクセスするための帯域幅が制限されるため、更新が完了する前にタイムアウトが発生する可能性があります。シングルノードの OpenShift クラスターでは、事前キャッシュを使用して、これを回避できます。preCaching フィールドを true に設定して ClusterGroupUpgrade CR を作成すると、コンテナーイメージの事前キャッシュが開始されます。TALM は、使用可能なディスク容量を OpenShift Container Platform イメージの推定サイズと比較して、十分な容量があることを確認します。クラスターに十分なスペースがない場合、TALM はそのクラスターの事前キャッシュをキャンセルし、そのクラスターのポリシーを修復しません。

TALM は PrecacheSpecValid 条件を使用して、次のようにステータス情報を報告します。

- true

  事前キャッシュの仕様は有効で一貫性があります。

- false

  事前キャッシュの仕様は不完全です。

TALM は PrecachingSucceeded 条件を使用して、次のようにステータス情報を報告します。

- true

  TALM は事前キャッシュプロセスを完了しました。いずれかのクラスターで事前キャッシュが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。クラスターの事前キャッシュが失敗した場合は、メッセージで通知されます。

- false

  1 つ以上のクラスターで事前キャッシュがまだ進行中か、すべてのクラスターで失敗しました。

詳細については、コンテナーイメージの事前キャッシュ機能の使用セクションを参照してください。

12.5.4. バックアップの作成

単一ノードの OpenShift の場合、TALM は更新前にデプロイメントのバックアップを作成できます。アップデートが失敗した場合は、以前のバージョンを回復し、アプリケーションの再プロビジョニングを必要とせずにクラスターを動作状態に復元できます。バックアップ機能を使用するには、最初に backup フィールドを true に設定して ClusterGroupUpgrade CR を作成します。バックアップの内容が最新であることを確認するために、ClusterGroupUpgrade CR の enable フィールドを true に設定するまで、バックアップは取得されません。

TALM は BackupSucceeded 条件を使用して、ステータスと理由を次のように報告します。

- true

  すべてのクラスターのバックアップが完了したか、バックアップの実行が完了したが、1 つ以上のクラスターで失敗しました。いずれかのクラスターのバックアップが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。

- false

  1 つ以上のクラスターのバックアップがまだ進行中か、すべてのクラスターのバックアップが失敗しました。

詳細については、アップグレード前のクラスターリソースのバックアップの作成セクションを参照してください。

12.5.5. クラスターの更新

TALM は、修復計画に従ってポリシーを適用します。以降のバッチに対するポリシーの適用は、現在のバッチのすべてのクラスターがすべての管理ポリシーに準拠した直後に開始されます。バッチがタイムアウトすると、TALM は次のバッチに移動します。バッチのタイムアウト値は、spec.timeout フィールドは修復計画のバッチ数で除算されます。

TALM は Progressing 条件を使用して、ステータスと理由を次のように報告します。

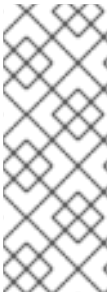
- true

  TALM は準拠していないポリシーを修正しています。

- false

  更新は進行中ではありません。これには次の理由が考えられます。

  - すべてのクラスターは、すべての管理ポリシーに準拠しています。
  - ポリシーの修復に時間がかかりすぎたため、更新がタイムアウトしました。
  - ブロッキング CR がシステムにないか、まだ完了していません。
  - ClusterGroupUpgrade CR が有効になっていません。
  - バックアップはまだ進行中です。



注記

管理されたポリシーは、ClusterGroupUpgrade CR の managedPolicies フィールドに一覧表示される順序で適用されます。1 つの管理ポリシーが一度に指定されたクラスターに適用されます。クラスターが現在のポリシーに準拠している場合、次の管理ポリシーがクラスターに適用されます。

Progressing 状態の ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion:
      deleteObjects: true
```

```
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: true
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 2
    timeout: 240
  clusterLabelSelectors:
    - matchExpressions:
      - key: label1
        operator: In
        values:
          - value1a
          - value1b
  batchTimeoutAction:
status:
  clusters:
    - name: spoke1
      state: complete
  computedMaxConcurrency: 2
  conditions:
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: All selected clusters are valid
      reason: ClusterSelectionCompleted
      status: 'True'
      type: ClustersSelected
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: Completed validation
      reason: ValidationCompleted
      status: 'True'
      type: Validated
    - lastTransitionTime: '2022-11-18T16:37:16Z'
      message: Remediating non-compliant policies
      reason: InProgress
      status: 'True'
      type: Progressing (1)
  managedPoliciesForUpgrade:
    - name: talm-policy
      namespace: talm-namespace
  managedPoliciesNs:
    talm-policy: talm-namespace
  remediationPlan:
    - - spoke1
    - - spoke2
      - spoke3
  status:
    currentBatch: 2
    currentBatchRemediationProgress:
      spoke2:
        state: Completed
```

```
    spoke3:
      policyIndex: 0
      state: InProgress
  currentBatchStartedAt: '2022-11-18T16:27:16Z'
  startedAt: '2022-11-18T16:27:15Z'
```

1

Progressing フィールドは、TALM がポリシーの修復中であることを示しています。

12.5.6. 更新ステータス

TALM は Succeeded 条件を使用して、ステータスと理由を次のようにレポートします。

- true

  すべてのクラスターは、指定された管理ポリシーに準拠しています。

- false

  修正に使用できるクラスターがないか、次のいずれかの理由でポリシーの修正に時間がかかりすぎたため、ポリシーの修正に失敗しました。

  - 現在のバッチにカナリア更新が含まれており、バッチ内のクラスターがバッチタイムアウト内のすべての管理ポリシーに準拠していません。
  - クラスターは、remediationStrategy フィールドに指定された timeout 値内で管理ポリシーに準拠していませんでした。

Succeeded 状態の ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
```

```
  namespace: default
spec:
  clusters:
  - spoke1
  - spoke4
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: 1
  clusters:
    - name: spoke1
      state: complete
    - name: spoke4
      state: complete
  conditions:
  - message: All selected clusters are valid
    reason: ClusterSelectionCompleted
    status: "True"
    type: ClustersSelected
  - message: Completed validation
    reason: ValidationCompleted
    status: "True"
    type: Validated
  - message: All clusters are compliant with all the managed policies
    reason: Completed
    status: "False"
    type: Progressing 2
  - message: All clusters are compliant with all the managed policies
    reason: Completed
    status: "True"
    type: Succeeded 3
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  remediationPlan:
  - - spoke1
  - - spoke4
  status:
    completedAt: '2022-11-18T16:27:16Z'
    startedAt: '2022-11-18T16:27:15Z'
```

2

Progressing フィールドでは、更新が完了したため、ステータスは false です。クラスターはすべての管理ポリシーに準拠しています。

3

Succeeded フィールドは、検証が正常に完了したことを示しています。

1

status フィールドには、クラスターのリストとそれぞれのステータスが含まれます。クラスターのステータスは、complete または timedout です。

タイムアウト 状態の ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
    - spoke2
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
status:
  clusters:
    - name: spoke1
      state: complete
    - currentPolicy: 1
        name: talm-policy
        status: NonCompliant
      name: spoke2
      state: timedout
  computedMaxConcurrency: 2
  conditions:
    - lastTransitionTime: '2022-11-18T16:27:15Z'
```

```
    message: All selected clusters are valid
    reason: ClusterSelectionCompleted
    status: 'True'
    type: ClustersSelected
  - lastTransitionTime: '2022-11-18T16:27:15Z'
    message: Completed validation
    reason: ValidationCompleted
    status: 'True'
    type: Validated
  - lastTransitionTime: '2022-11-18T16:37:16Z'
    message: Policy remediation took too long
    reason: TimedOut
    status: 'False'
    type: Progressing
  - lastTransitionTime: '2022-11-18T16:37:16Z'
    message: Policy remediation took too long
    reason: TimedOut
    status: 'False'
    type: Succeeded 2
managedPoliciesForUpgrade:
  - name: talm-policy
    namespace: talm-namespace
managedPoliciesNs:
  talm-policy: talm-namespace
remediationPlan:
  - - spoke1
    - spoke2
status:
    startedAt: '2022-11-18T16:27:15Z'
    completedAt: '2022-11-18T20:27:15Z'
```

1

クラスターの状態が timedout の場合、currentPolicy フィールドにはポリシーの名前とポリシーのステータスが表示されます。

2

succeeded のステータスは false であり、ポリシーの修正に時間がかかりすぎたことを示すメッセージが表示されます。

12.5.7. ClusterGroupUpgrade CR のブロック

複数の ClusterGroupUpgrade CR を作成して、それらの適用順序を制御できます。

たとえば、ClusterGroupUpgrade CR A の開始をブロックする ClusterGroupUpgrade CR C を作

成する場合、ClusterGroupUpgrade CR A は ClusterGroupUpgrade CR C のステータスが UpgradeComplete になるまで起動できません。

1 つの ClusterGroupUpgrade CR には複数のブロッキング CR を含めることができます。この場合、現在の CR のアップグレードを開始する前に、すべてのブロッキング CR を完了する必要があります。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. ClusterGroupUpgrade CR の内容を cgu-a.yaml、cgu-b.yaml、および cgu-c.yaml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
```

```
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
```

1

ブロッキング CR を定義します。cgu-c が完了するまで cgu-a の更新を開始できません。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
```

```
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

1

cgu-a が完了するまで cgu-b の更新を開始できません。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec: 1
  clusters:
  - spoke6
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
```

```
    status: "False"
    type: Ready
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status: {}
```

1

cgu-c の更新にはブロック CR がありません。TALM は、enable フィールドが true に設定されている場合に cgu-c の更新を開始します。

2. 関連する CR ごとに以下のコマンドを実行して ClusterGroupUpgrade CR を作成します。

```
$ oc apply -f <name>.yaml
```

3. 関連する各 CR について以下のコマンドを実行して、更新プロセスを開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/<name> \
--type merge -p '{"spec":{"enable":true}}'
```

以下の例は、enable フィールドが true に設定されている ClusterGroupUpgrade CR を示しています。

ブロッキング CR のある cgu-a の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
```

```
  blockingCRs:
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-c]' 1
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
  status: {}
```

1

ブロッキング CR のリストを表示します。

ブロッキング CR のある cgu-b の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs:
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-a]' (1)
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

1

ブロッキング CR のリストを表示します。

CR をブロックする cgu-c の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec:
  clusters:
  - spoke6
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant 1
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status:
```

```
currentBatch: 1
remediationPlanForBatch:
  spoke6: 0
```

1

cgu-c の更新にはブロック CR がありません。

12.6. マネージドクラスターでのポリシーの更新

Topology Aware Lifecycle Manager (TALM) は、ClusterGroupUpgrade CR で指定されたクラスターの inform ポリシーのセットを修正します。TALM は、PlacementBinding CR の bindingOverrides. remediationAction および subFilter 仕様を使用して、Policy CR の remediationAction 仕様を制御することにより、inform ポリシーを修正します。コピーされた各ポリシーには、それぞれの対応する RHACM 配置ルールと RHACM 配置バインディングがあります。

1 つずつ、TALM は、現在のバッチから、適用可能な管理ポリシーに対応する配置ルールに各クラスターを追加します。クラスターがポリシーにすでに準拠している場合は、TALM は準拠するクラスターへのポリシーの適用を省略します。次に TALM は次のポリシーを非準拠クラスターに適用します。TALM がバッチの更新を完了すると、コピーしたポリシーに関連付けられた配置ルールからすべてのクラスターが削除されます。次に、次のバッチの更新が開始されます。

スポーククラスターの状態が RHACM に準拠している状態を報告しない場合、ハブクラスターの管理ポリシーには TALM が必要とするステータス情報がありません。TALM は、以下の方法でこれらのケースを処理します。

- ポリシーの status.compliant フィールドがない場合、TALM はポリシーを無視してログエントリーを追加します。次に、TALM はポリシーの status.status フィールドを確認し続けます。
- ポリシーの status.status がない場合、TALM はエラーを生成します。
- クラスターのコンプライアンスステータスがポリシーの status.status フィールドにない場合、TALM はそのクラスターをそのポリシーに準拠していないと見なします。

ClusterGroupUpgrade CR の batchTimeoutAction は、クラスターのアップグレードが失敗した場合にどうなるかを決定します。continue を指定して失敗したクラスターをスキップし、他のクラス

ターのアップグレードを続行するか、abort を指定してすべてのクラスターのポリシー修正を停止することができます。タイムアウトが経過すると、TALM は作成したすべてのリソースを削除して、クラスターにこれ以上更新が行われないようにします。

アップグレードポリシーの例

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: ocp-4.4.16.4
  namespace: platform-upgrade
spec:
  disabled: false
  policy-templates:
  - objectDefinition:
      apiVersion: policy.open-cluster-management.io/v1
      kind: ConfigurationPolicy
      metadata:
        name: upgrade
      spec:
        namespaceselector:
          exclude:
          - kube-*
          include:
          - '*'
        object-templates:
        - complianceType: musthave
          objectDefinition:
            apiVersion: config.openshift.io/v1
            kind: ClusterVersion
            metadata:
              name: version
            spec:
              channel: stable-4.16
              desiredUpdate:
                version: 4.4.16.4
              upstream: https://api.openshift.com/api/upgrades_info/v1/graph
            status:
              history:
                - state: Completed
                  version: 4.4.16.4
        remediationAction: inform
        severity: low
  remediationAction: inform
```

RHACM ポリシーの詳細は、ポリシーの概要 を参照してください。

関連情報

- PolicyGenerator CRD について

12.6.1. TALM を使用してインストールするマネージドクラスターの Operator サブスクリプションの設定

Topology Aware Lifecycle Manager (TALM) は、Operator の Subscription カスタムリソース (CR) に status.state.AtlatestKnown フィールドが含まれている場合に限り、Operator のインストールプランを承認できます。

手順

1. Operator の Subscription CR に、status.state.AtlatestKnown フィールドを追加します。

Subscription CR の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: "stable"
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown (1)
```

(1) status.state: AtlatestKnown フィールドは、Operator カタログから入手可能な Operator の最新バージョンに使用されます。

注記

新しいバージョンの Operator がレジストリーで利用可能になると、関連するポリシーが非準拠になります。

2. ClusterGroupUpgrade CR を使用して、変更した Subscription ポリシーをマネージドクラスターに適用します。

12.6.2. マネージドクラスターへの更新ポリシーの適用

ポリシーを適用してマネージドクラスターを更新できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- TALM 4.16 には RHACM 2.9 以降が必要です。
- 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. ClusterGroupUpgrade CR の内容を cgu-1.yaml ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
  managedPolicies: 1
    - policy1-common-cluster-version-policy
    - policy2-common-nto-sub-policy
```

```
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
enable: false
clusters: 2
- spoke1
- spoke2
- spoke5
- spoke6
remediationStrategy:
  maxConcurrency: 2 3
  timeout: 240 4
batchTimeoutAction: 5
```

1 適用するポリシーの名前。

2 更新するクラスターのリスト。

3 maxConcurrency フィールドは、同時に更新されるクラスターの数を示します。

4 更新のタイムアウト (分単位)。

5 バッチがタイムアウトした場合の動作を制御します。可能な値は abort または continue です。指定しない場合、デフォルトは continue です。

2. 以下のコマンドを実行して ClusterGroupUpgrade CR を作成します。

```
$ oc create -f cgu-1.yaml
```

a. 以下のコマンドを実行して、ClusterGroupUpgrade CR がハブクラスターに作成されていることを確認します。

```
$ oc get cgu --all-namespaces
```

出力例

```
NAMESPACE   NAME  AGE  STATE      DETAILS
default     cgu-1 8m55 NotEnabled Not Enabled
```

b. 以下のコマンドを実行して更新のステータスを確認します。

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

出力例

```
{
  "computedMaxConcurrency": 2,
  "conditions": [
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "Not enabled", 1
      "reason": "NotEnabled",
      "status": "False",
      "type": "Progressing"
    }
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-
tuning-operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-
operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "
[{\"kind\":\"Subscription\",\"name\":\"sriov-network-operator-
subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-nto-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
```

```
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {}
}
```

1 ClusterGroupUpgrade CR の spec.enable フィールドは false に設定されます。

3. 以下のコマンドを実行して、spec.enable フィールドの値を true に変更します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-1 \
--patch '{"spec":{"enable":true}}' --type=merge
```

検証

1. 以下のコマンドを実行して更新のステータスを確認します。

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

出力例

```
{
 "computedMaxConcurrency": 2,
 "conditions": [ 1
  {
   "lastTransitionTime": "2022-02-25T15:33:07Z",
   "message": "All selected clusters are valid",
   "reason": "ClusterSelectionCompleted",
   "status": "True",
   "type": "ClustersSelected"
  },
  {
   "lastTransitionTime": "2022-02-25T15:33:07Z",
   "message": "Completed validation",
   "reason": "ValidationCompleted",
   "status": "True",
   "type": "Validated"
  },
  {
   "lastTransitionTime": "2022-02-25T15:34:07Z",
   "message": "Remediating non-compliant policies",
   "reason": "InProgress",
   "status": "True",
   "type": "Progressing"
  }
 ],
 "managedPoliciesContent": {
  "policy1-common-cluster-version-policy": "null",
  "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-
tuning-operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
  "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-
operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
  "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-
network-operator-subscription\",\"namespace\":\"openshift-sriov-network-
operator\"}]"
 },
 "managedPoliciesForUpgrade": [
  {
   "name": "policy1-common-cluster-version-policy",
   "namespace": "default"
  },
  {
   "name": "policy2-common-nto-sub-policy",
   "namespace": "default"
  },
  {
```

```
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {
    "currentBatch": 1,
    "currentBatchRemediationProgress": {
      "spoke1": {
        "policyIndex": 1,
        "state": "InProgress"
      },
      "spoke2": {
        "policyIndex": 1,
        "state": "InProgress"
      }
    },
    "currentBatchStartedAt": "2022-02-25T15:54:16Z",
    "startedAt": "2022-02-25T15:54:16Z"
  }
}
```

1

現在のバッチの更新の進捗を反映します。このコマンドを再度実行して、進捗に関する更新情報を取得します。

2. 以下のコマンドを実行してポリシーのステータスを確認します。

```
oc get policies -A
```

出力例

```
NAMESPACE   NAME                                        REMEDIATION ACTION   COMPLIANCE
STATE     AGE
spoke1    default.policy1-common-cluster-version-policy enforce            Compliant
18m
spoke1    default.policy2-common-nto-sub-policy         enforce            NonCompliant
18m
spoke2    default.policy1-common-cluster-version-policy enforce            Compliant
18m
spoke2    default.policy2-common-nto-sub-policy         enforce            NonCompliant
18m
spoke5    default.policy3-common-ptp-sub-policy         inform             NonCompliant
18m
spoke5    default.policy4-common-sriov-sub-policy       inform             NonCompliant
18m
spoke6    default.policy3-common-ptp-sub-policy         inform             NonCompliant
18m
spoke6    default.policy4-common-sriov-sub-policy       inform             NonCompliant
18m
default   policy1-common-ptp-sub-policy                 inform             Compliant
18m
default   policy2-common-sriov-sub-policy               inform             NonCompliant
18m
default   policy3-common-ptp-sub-policy                 inform             NonCompliant
18m
default   policy4-common-sriov-sub-policy               inform             NonCompliant
18m
```

- spec.remediationAction 値が、現在のバッチからクラスターに適用される子ポリシーの enforce に変更されます。
- spec.remedationAction 値は、残りのクラスターの子ポリシーの inform を維持しま

す。

- バッチが完了すると、spec.remediationAction 値が変更され、強制された子ポリシーの inform に戻ります。

3. ポリシーに Operator サブスクリプションが含まれる場合、インストールの進捗を単一ノードクラスターで直接確認できます。

a. 以下のコマンドを実行して、インストールの進捗を確認する単一ノードクラスターの KUBECONFIG ファイルをエクスポートします。

```
$ export KUBECONFIG=<cluster_kubeconfig_absolute_path>
```

b. 単一ノードクラスターに存在するすべてのサブスクリプションを確認し、以下のコマンドを実行し、ClusterGroupUpgrade CR でインストールしようとしているポリシーを探します。

```
$ oc get subs -A | grep -i <subscription_name>
```

cluster-logging ポリシーの出力例

```
NAMESPACE                    NAME                 PACKAGE
SOURCE            CHANNEL
openshift-logging            cluster-logging      cluster-logging
redhat-operators   stable
```

4. 管理ポリシーの 1 つに ClusterVersion CR が含まれる場合は、スポーククラスターに対して以下のコマンドを実行して、現在のバッチでプラットフォーム更新のステータスを確認します。

```
$ oc get clusterversion
```

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.4.16.5  True        True          43s     Working towards 4.4.16.7: 71 of 735 done
(9% complete)
```

5. 以下のコマンドを実行して Operator サブスクリプションを確認します。

```
$ oc get subs -n <operator-namespace> <operator-subscription> -ojsonpath="
{.status}"
```

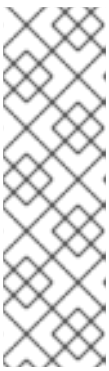
6. 以下のコマンドを実行して、必要なサブスクリプションに関連付けられている単一ノードのクラスターに存在するインストール計画を確認します。

```
$ oc get installplan -n <subscription_namespace>
```

cluster-logging Operator の出力例

```
NAMESPACE                    NAME            CSV                        APPROVAL
APPROVED
openshift-logging            install-6khtw   cluster-logging.5.3.3-4    Manual
true 1
```

1. インストール計画の Approval フィールドは Manual に設定されており、TALM がインストール計画を承認すると、Approved フィールドは false から true に変わります。



注記

TALM がサブスクリプションを含むポリシーを修正している場合、そのサブスクリプションに関連付けられているすべてのインストールプランが自動的に承認されます。オペレーターが最新の既知のバージョンに到達するために複数のインストールプランが必要な場合、TALM は複数のインストールプランを承認し、最終バージョンに到達するために 1 つ以上の中間バージョンをアップグレードします。

7. 以下のコマンドを実行して、ClusterGroupUpgrade がインストールしているポリシーの Operator のクラスターサービスバージョンが Succeeded フェーズに到達したかどうかを確認します。

```
$ oc get csv -n <operator_namespace>
```

OpenShift Logging Operator の出力例

```
NAME                  DISPLAY                  VERSION   REPLACES   PHASE
cluster-logging.5.4.2   Red Hat OpenShift Logging   5.4.2               Succeeded
```

12.7. アップグレード前のクラスターリソースのバックアップの作成

単一ノードの OpenShift の場合、Topology Aware Lifecycle Manager (TALM) は、アップグレード前にデプロイメントのバックアップを作成できます。アップグレードが失敗した場合は、以前のバージョンを回復し、アプリケーションの再プロビジョニングを必要とせずにクラスターを動作状態に復元できます。

バックアップ機能を使用するには、最初に backup フィールドを true に設定して ClusterGroupUpgrade CR を作成します。バックアップの内容が最新であることを確認するために、ClusterGroupUpgrade CR の enable フィールドを true に設定するまで、バックアップは取得されません。

TALM は BackupSucceeded 条件を使用して、ステータスと理由を次のように報告します。

- true

  すべてのクラスターのバックアップが完了したか、バックアップの実行が完了したが、1 つ以上のクラスターで失敗しました。いずれかのクラスターでバックアップが失敗した場合、そのクラスターの更新は続行されません。

- false

  1 つ以上のクラスターのバックアップがまだ進行中か、すべてのクラスターのバックアップが失敗しました。スポーククラスターで実行されているバックアッププロセスには、次のス

テータスがあります。

- PreparingToStart

  最初の調整パスが進行中です。TALM は、失敗したアップグレード試行で作成されたスポークバックアップネームスペースとハブビューリソースをすべて削除します。

- Starting

  バックアップの前提条件とバックアップジョブを作成しています。

- Active

  バックアップが進行中です。

- Succeeded

  バックアップは成功しました。

- BackupTimeout

  アーティファクトのバックアップは部分的に行われます。

- UnrecoverableError

  バックアップはゼロ以外の終了コードで終了しました。

注記

クラスターのバックアップが失敗し、BackupTimeout または UnrecoverableError 状態になると、そのクラスターのクラスター更新は続行されません。他のクラスターへの更新は影響を受けず、続行されます。

12.7.1. バックアップを含む ClusterGroupUpgrade CR の作成

シングルノードの OpenShift クラスターでアップグレードする前に、デプロイメントのバックアップを作成できます。アップグレードが失敗した場合は、Topology Aware Lifecycle Manager (TALM) によって生成された upgrade-recovery.sh スクリプトを使用して、システムをアップグレード前の状態に戻すことができます。バックアップは次の項目で設定されます。

クラスターのバックアップ

: etcd と静的 Pod マニフェストのスナップショット。

コンテンツのバックアップ

: /etc、/usr/local、/var/lib/kubelet などのフォルダーのバックアップ。

変更されたファイルのバックアップ

: 変更された machine-config によって管理されるすべてのファイル。

Deployment

: 固定された ostree デプロイメント。

イメージ (オプション)

: 使用中のコンテナーイメージ。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- Red Hat Advanced Cluster Management 2.2.4 をインストールします。

注記

リカバリーパーティションを作成することを強く推奨します。以下は、50 GB のリカバリーパーティションの SiteConfig カスタムリソース (CR) の例です。

```
nodes:
   - hostName: "node-1.example.com"
   role: "master"
   rootDeviceHints:
      hctl: "0:2:0:0"
      deviceName: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
...
   #Disk /dev/disk/by-id/scsi-3600508b400105e210000900000490000:
   #893.3 GiB, 959119884288 bytes, 1873281024 sectors
   diskPartition:
      - device: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
      partitions:
      - mount_point: /var/recovery
         size: 51200
         start: 800000
```

手順

1. clustergroupupgrades-group-du.yaml ファイルで、backup フィールドと enable フィールドを true に設定して、ClusterGroupUpgrade CR の内容を保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true
  backup: true
  clusters:
  - cnfdb1
  - cnfdb2
  enable: true
  managedPolicies:
  - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

2. 更新を開始するには、次のコマンドを実行して ClusterGroupUpgrade CR を適用します。

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

検証

- 以下のコマンドを実行して、ハブクラスターのアップグレードのステータスを確認します。

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

出力例

```
{
   "backup": {
      "clusters": [
         "cnfdb2",
         "cnfdb1"
   ],
   "status": {
      "cnfdb1": "Succeeded",
      "cnfdb2": "Failed" 1
   }
},
"computedMaxConcurrency": 1,
"conditions": [
   {
      "lastTransitionTime": "2022-04-05T10:37:19Z",
      "message": "Backup failed for 1 cluster", 2
      "reason": "PartiallyDone", 3
      "status": "True", 4
      "type": "Succeeded"
   }
],
"precaching": {
   "spec": {}
},
"status": {}
```

1 1 つのクラスターのバックアップが失敗しました。

2 このメッセージは、1 つのクラスターのバックアップが失敗したことを確認します。

3

バックアップは部分的に成功しました。

4

バックアッププロセスが終了しました。

12.7.2. アップグレードが失敗した後のクラスターのリカバリー

クラスターのアップグレードが失敗した場合は、手動でクラスターにログインし、バックアップを使用してクラスターをアップグレード前の状態に戻すことができます。次の 2 つの段階があります。

ロールバック

試行されたアップグレードにプラットフォーム OS 展開への変更が含まれていた場合は、回復スクリプトを実行する前に、以前のバージョンにロールバックする必要があります。

重要

ロールバックは、TALM および単一ノード OpenShift からのアップグレードにのみ適用されます。このプロセスは、他のアップグレードタイプからのロールバックには適用されません。

復元

リカバリーはコンテナーをシャットダウンし、バックアップパーティションのファイルを使用してコンテナーを再起動し、クラスターを復元します。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1 つ以上のマネージドクラスターをプロビジョニングします。
- Red Hat Advanced Cluster Management 2.2.4 をインストールします。
- cluster-admin 権限を持つユーザーとしてログインしている。

- バックアップ用に設定されたアップグレードを実行します。

手順

1. 次のコマンドを実行して、以前に作成した ClusterGroupUpgrade カスタムリソース (CR) を削除します。

```
$ oc delete cgu/du-upgrade-4918 -n ztp-group-du-sno
```

2. リカバリーするクラスターにログインします。

3. 次のコマンドを実行して、プラットフォーム OS の展開のステータスを確認します。

```
$ ostree admin status
```

出力例

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9.0
    Version: 49.84.202202230006-0
    Pinned: yes (1)
    origin refspec:
c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9
```

(1) 現在の展開は固定されています。プラットフォーム OS 展開のロールバックは必要ありません。

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa.0
    Version: 410.84.202204050541-0
    origin refspec:
f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa
rhcos ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca.0
(rollback) (1)
    Version: 410.84.202203290245-0
```

```
    Pinned: yes (2)
    origin refspec:
ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca
```

(1) このプラットフォーム OS の展開は、ロールバックの対象としてマークされています。

(2) 以前の展開は固定されており、ロールバックできます。

4. プラットフォーム OS 展開のロールバックをトリガーするには、次のコマンドを実行します。

```
$ rpm-ostree rollback -r
```

5. 復元の最初のフェーズでは、コンテナーをシャットダウンし、ファイルをバックアップパーティションから対象のディレクトリーに復元します。リカバリーを開始するには、次のコマンドを実行します。

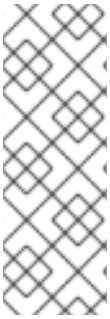
```
$ /var/recovery/upgrade-recovery.sh
```

6. プロンプトが表示されたら、次のコマンドを実行してクラスターを再起動します。

```
$ systemctl reboot
```

7. 再起動後、次のコマンドを実行してリカバリーを再開します。

```
$ /var/recovery/upgrade-recovery.sh  --resume
```



注記

リカバリーユーティリティーが失敗した場合は、--restart オプションを使用して再試行できます。

```
$ /var/recovery/upgrade-recovery.sh --restart
```

検証

- リカバリーのステータスを確認するには、次のコマンドを実行します。

```
$ oc get clusterversion,nodes,clusteroperator
```

出力例

```
NAME                                         VERSION   AVAILABLE   PROGRESSING   SINCE
STATUS
clusterversion.config.openshift.io/version   4.4.16.23    True        False         86d
Cluster version is 4.4.16.23 (1)


NAME                        STATUS   ROLES          AGE   VERSION
node/lab-test-spoke1-node-0   Ready    master,worker   86d   v1.22.3+b93fd35 (2)

NAME                                                       VERSION   AVAILABLE
PROGRESSING   DEGRADED   SINCE   MESSAGE
clusteroperator.config.openshift.io/authentication                    4.4.16.23    True
False         False      2d7h (3)
clusteroperator.config.openshift.io/baremetal                         4.4.16.23    True
False         False      86d


.............
```

(1) クラスターのバージョンが利用可能であり、正しいバージョンを持っています。

(2) ノードのステータスは Ready です。

(3) ClusterOperator オブジェクトの可用性は True です。

12.8. コンテナーイメージ事前キャッシュ機能の使用

シングルノードの OpenShift クラスターでは、コンテナーイメージレジストリーにアクセスするための帯域幅が制限されている可能性があり、更新が完了する前に、タイムアウトが発生する可能性があ

ります。

注記

更新の時間は TALM によって設定されていません。手動アプリケーションまたは外部自動化により、更新の開始時に ClusterGroupUpgrade CR を適用できます。

コンテナーイメージの事前キャッシュは、ClusterGroupUpgrade CR で preCaching フィールドが true に設定されている場合に起動します。

TALM は PrecacheSpecValid 条件を使用して、次のようにステータス情報を報告します。

- true

  事前キャッシュの仕様は有効で一貫性があります。

- false

  事前キャッシュの仕様は不完全です。

TALM は PrecachingSucceeded 条件を使用して、次のようにステータス情報を報告します。

- true

  TALM は事前キャッシュプロセスを完了しました。いずれかのクラスターで事前キャッシュが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。クラスターの事前キャッシュが失敗した場合は、メッセージで通知されます。

- false

  1 つ以上のクラスターで事前キャッシュがまだ進行中か、すべてのクラスターで失敗しました。

事前キャッシュプロセスに成功すると、ポリシーの修正を開始できます。修復アクションは、enable フィールドが true に設定されている場合に開始されます。クラスターで事前キャッシュエラーが発生した場合、そのクラスターのアップグレードは失敗します。アップグレードプロセスは、事前キャッシュが成功した他のすべてのクラスターに対して続行されます。

事前キャッシュプロセスは、以下のステータスにあります。

- NotStarted

  これは、すべてのクラスターが ClusterGroupUpgrade CR の最初の調整パスで自動的に割り当てられる初期状態です。この状態では、TALM は、以前の不完全な更新から残ったスポーククラスターの事前キャッシュの namespace およびハブビューリソースを削除します。次に TALM は、スポーク前の namespace の新規の ManagedClusterView リソースを作成し、PrecachePreparing 状態の削除を確認します。

- PreparingToStart

  以前の不完全な更新からの残りのリソースを消去すると進行中です。

- Starting

  キャッシュ前のジョブの前提条件およびジョブが作成されます。

- Active

  ジョブは Active の状態です。

- Succeeded

  事前キャッシュジョブが成功しました。

- PrecacheTimeout

  アーティファクトの事前キャッシュは部分的に行われます。

- UnrecoverableError

  ジョブはゼロ以外の終了コードで終了します。

12.8.1. コンテナーイメージの事前キャッシュフィルターの使用

通常、事前キャッシュ機能は、クラスターが更新に必要とするよりも多くのイメージをダウンロードします。どの事前キャッシュイメージをクラスターにダウンロードするかを制御できます。これにより、ダウンロード時間が短縮され、帯域幅とストレージが節約されます。

次のコマンドを使用して、ダウンロードするすべてのイメージのリストを表示できます。

```
$ oc adm release info <ocp-version>
```

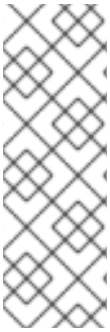
次の ConfigMap の例は、excludePrecachePatterns フィールドを使用してイメージを除外する方法を示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-group-upgrade-overrides
data:
  excludePrecachePatterns: |
    azure (1)
    aws
    vsphere
    alibaba
```

(1) TALM は、ここにリストされているパターンのいずれかを含む名前を持つすべてのイメージを除外します。

12.8.2. 事前キャッシュでの ClusterGroupUpgrade CR の作成

シングルノードの OpenShift の場合は、事前キャッシュ機能により、更新が開始する前に、必要なコンテナーイメージをスポーククラスターに配置できます。



注記

事前キャッシュの場合、TALM は ClusterGroupUpgrade CR の spec.remediationStrategy.timeout 値を使用します。事前キャッシュジョブが完了するのに十分な時間を与える timeout 値を設定する必要があります。事前キャッシュの完了後に ClusterGroupUpgrade CR を有効にすると、timeout 値を更新に適した期間に変更できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. clustergroupupgrades-group-du.yaml ファイルで preCaching フィールドを true に設定して ClusterGroupUpgrade CR の内容を保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true (1)
  clusters:
  - cnfdb1
  - cnfdb2
  enable: false
  managedPolicies:
  - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

(1) preCaching フィールドは true に設定されています。これにより、更新を開始する前に TALM がコンテナーイメージをプルできます。

2.

事前キャッシュを開始する場合は、次のコマンドを実行して ClusterGroupUpgrade CR を適用します。

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

検証

1. 以下のコマンドを実行して、ClusterGroupUpgrade CR がハブクラスターに存在するかどうかを確認します。

```
$ oc get cgu -A
```

出力例

```
NAMESPACE          NAME              AGE   STATE       DETAILS
ztp-group-du-sno   du-upgrade-4918   10s   InProgress   Precaching is required and not done 1
```

1 CR が作成されます。

2. 以下のコマンドを実行して、事前キャッシュタスクのステータスを確認します。

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

出力例

```
{
 "conditions": [
   {
    "lastTransitionTime": "2022-01-27T19:07:24Z",
    "message": "Precaching is required and not done",
    "reason": "InProgress",
    "status": "False",
    "type": "PrecachingSucceeded"
   },
```

```
    {
     "lastTransitionTime": "2022-01-27T19:07:34Z",
     "message": "Pre-caching spec is valid and consistent",
     "reason": "PrecacheSpecIsWellFormed",
     "status": "True",
     "type": "PrecacheSpecValid"
    }
   ],
   "precaching": {
    "clusters": [
     "cnfdb1" 1
     "cnfdb2"
    ],
    "spec": {
     "platformImage": "image.example.io"},
    "status": {
     "cnfdb1": "Active"
     "cnfdb2": "Succeeded"}
    }
}
```

1

特定されたクラスターの一覧を表示します。

3. スポーククラスターで以下のコマンドを実行して、事前キャッシュジョブのステータスを確認します。

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

出力例

```
NAME                COMPLETIONS   DURATION   AGE
job.batch/pre-cache   0/1           3m10s      3m10s

NAME                      READY   STATUS    RESTARTS   AGE
pod/pre-cache--1-9bmlr    1/1     Running   0          3m10s
```

4. 以下のコマンドを実行して ClusterGroupUpgrade CR のステータスを確認します。

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

出力例

```
"conditions": [
   {
    "lastTransitionTime": "2022-01-27T19:30:41Z",
    "message": "The ClusterGroupUpgrade CR has all clusters compliant with all the
managed policies",
    "reason": "UpgradeCompleted",
    "status": "True",
    "type": "Ready"
   },
   {
    "lastTransitionTime": "2022-01-27T19:28:57Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
    "type": "PrecachingSucceeded" 1
   }
```

1

キャッシュ前のタスクが実行されます。

12.9. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング

Topology Aware Lifecycle Manager (TALM) は、RHACM ポリシーを修復する OpenShift Container Platform Operator です。問題が発生した場合には、oc adm must-gather コマンドを使用して詳細およびログを収集し、問題のデバッグ手順を行います。

関連トピックの詳細は、以下のドキュメントを参照してください。

- Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix
- Red Hat Advanced Cluster Management Troubleshooting

- Operator の問題のトラブルシューティングセクション

12.9.1. 一般的なトラブルシューティング

以下の質問を確認して、問題の原因を特定できます。

- 適用する設定がサポートされているか？
  - RHACM と OpenShift Container Platform のバージョンと互換性があるか？
  - TALM および RHACM のバージョンと互換性があるか？
- 問題の原因となる以下のコンポーネントはどれですか？
  - 「管理ポリシー」
  - 「クラスター」
  - 「修復ストラテジー」
  - 「Topology Aware Lifecycle Manager」

ClusterGroupUpgrade 設定が機能するようにするには、以下を実行できます。

1. spec.enable フィールドを false に設定して ClusterGroupUpgrade CR を作成します。
2. ステータスが更新され、トラブルシューティングの質問を確認するのを待ちます。
3. すべてが予想通りに機能する場合は、ClusterGroupUpgrade CR で spec.enable フィールドを true に設定します。

警告

ClusterUpgradeGroup CR で spec.enable フィールドを true に設定すると、更新手順が起動し、CR の spec フィールドを編集することができなくなります。

12.9.2. ClusterUpgradeGroup CR を変更できません。

問題

更新を有効にした後に、ClusterUpgradeGroup CR を編集することはできません。

解決方法

以下の手順を実行して手順を再起動します。

1. 以下のコマンドを実行して古い ClusterGroupUpgrade CR を削除します。

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. マネージドクラスターおよびポリシーに関する既存の問題を確認し、修正します。

   a. すべてのクラスターがマネージドクラスターで、利用可能であることを確認します。

   b. すべてのポリシーが存在し、spec.remediationAction フィールドが inform に設定されていることを確認します。

3. 正しい設定で新規の ClusterGroupUpgrade CR を作成します。

```
$ oc apply -f <ClusterGroupUpgradeCR_YAML>
```

12.9.3. 管理ポリシー

システムでの管理ポリシーの確認

問題

システムで正しい管理ポリシーがあるかどうかをチェックする。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

出力例

```
["group-du-sno-validator-du-validator-policy", "policy2-common-nto-sub-policy", "policy3-common-ptp-sub-policy"]
```

remediationAction モードの確認

問題

remediationAction フィールドが、管理ポリシーの spec で inform に設定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get policies --all-namespaces
```

出力例

```
NAMESPACE   NAME                                REMEDIATION ACTION   COMPLIANCE STATE   AGE
default     policy1-common-cluster-version-policy          inform           NonCompliant
5d21h
default     policy2-common-nto-sub-policy                  inform           Compliant
5d21h
default     policy3-common-ptp-sub-policy                  inform           NonCompliant
```

```
5d21h
default    policy4-common-sriov-sub-policy                inform           NonCompliant
5d21h
```

ポリシーコンプライアンスの状態の確認

問題

ポリシーのコンプライアンス状態を確認する。

解決方法

以下のコマンドを実行します。

```
$ oc get policies --all-namespaces
```

出力例

```
NAMESPACE   NAME                                REMEDIATION ACTION   COMPLIANCE
STATE   AGE
default    policy1-common-cluster-version-policy          inform           NonCompliant
5d21h
default    policy2-common-nto-sub-policy                  inform           Compliant
5d21h
default    policy3-common-ptp-sub-policy                  inform           NonCompliant
5d21h
default    policy4-common-sriov-sub-policy                inform           NonCompliant
5d21h
```

12.9.4. クラスター

マネージドクラスターが存在するかどうかの確認

問題

ClusterGroupUpgrade CR のクラスターがマネージドクラスターかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.example.com:6443      True     Unknown     13d
spoke1          true           https://api.spoke1.example.com:6443   True     True        13d
spoke3          true           https://api.spoke3.example.com:6443   True     True        27h
```

1. または、TALM マネージャーログを確認します。

   a. 以下のコマンドを実行して、TALM マネージャーの名前を取得します。

   ```
   $ oc get pod -n openshift-operators
   ```

   出力例

   ```
   NAME                                         READY   STATUS    RESTARTS   AGE
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp   2/2     Running
   0          45m
   ```

   b. 以下のコマンドを実行して、TALM マネージャーログを確認します。

   ```
   $ oc logs -n openshift-operators \
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
   ```

   出力例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler
error {"reconciler group": "ran.openshift.io", "reconciler kind":
"ClusterGroupUpgrade", "name": "lab-upgrade", "namespace": "default",
"error": "Cluster spoke5555 is not a ManagedCluster"} (1)
sigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).processNextWorkItem
```

(1) エラーメッセージには、クラスターがマネージドクラスターではないことが分かります。

マネージドクラスターが利用可能かどうかの確認

問題

ClusterGroupUpgrade CR で指定されたマネージドクラスターが利用可能かどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED
AVAILABLE   AGE
local-cluster   true           https://api.hub.testlab.com:6443      True
Unknown     13d
spoke1          true           https://api.spoke1.testlab.com:6443   True
True        13d (1)
spoke3          true           https://api.spoke3.testlab.com:6443   True
True        27h (2)
```

(1) (2) マネージドクラスターの AVAILABLE フィールドの値は True です。

clusterLabelSelector のチェック

問題

ClusterGroupUpgrade CR で指定された clusterLabelSelector フィールドが、管理対象クラスターの少なくとも 1 つと一致するかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get managedcluster --selector=upgrade=true 1
```

1

更新するクラスターのラベルは upgrade:true です。

出力例

```
NAME          HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED   AVAILABLE   AGE
spoke1        true           https://api.spoke1.testlab.com:6443   True     True        13d
spoke3        true           https://api.spoke3.testlab.com:6443   True     True        27h
```

カナリアクラスターが存在するかどうかの確認

問題

カナリアクラスターがクラスターのリストに存在するかどうかを確認します。

ClusterGroupUpgrade CR の例

```
spec:
  remediationStrategy:
    canaries:
    - spoke3
    maxConcurrency: 2
    timeout: 240
```

```
  clusterLabelSelectors:
   - matchLabels:
       upgrade: true
```

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.clusters}'
```

出力例

```
["spoke1", "spoke3"]
```

1. 以下のコマンドを実行して、カナリアクラスターが clusterLabelSelector ラベルに一致するクラスターの一覧に存在するかどうかを確認します。

```
$ oc get managedcluster --selector=upgrade=true
```

出力例

```
NAME      HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE   AGE
spoke1    true           https://api.spoke1.testlab.com:6443   True     True      13d
spoke3    true           https://api.spoke3.testlab.com:6443   True     True      27h
```

注記

クラスターは、spec.clusters に存在し、spec.clusterLabelSelector ラベルによって一致する場合もあります。

スポーククラスターでの事前キャッシュステータスの確認

1. スポーククラスターで以下のコマンドを実行して、事前キャッシュのステータスを確認します。

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

12.9.5. 修復ストラテジー

remediationStrategy が ClusterGroupUpgrade CR に存在するかどうかの確認

問題

remediationStrategy が ClusterGroupUpgrade CR に存在するかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

出力例

```
{"maxConcurrency":2, "timeout":240}
```

ClusterGroupUpgrade CR に maxConcurrency が指定されているかどうかの確認

問題

maxConcurrency が ClusterGroupUpgrade CR で指定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy.maxConcurrency}'
```

出力例

```
2
```

12.9.6. Topology Aware Lifecycle Manager

ClusterGroupUpgrade CR での条件メッセージおよびステータスの確認

問題

ClusterGroupUpgrade CR の status.conditions フィールドの値を確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

出力例

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"Missing managed policies:[policyList]", "reason":"NotAllManagedPoliciesExist", "status":"False", "type":"Validated"}
```

status.remediationPlan が計算されたかどうかの確認

問題

status.remediationPlan が計算されているかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.remediationPlan}'
```

出力例

```
[["spoke2", "spoke3"]]
```

TALM マネージャーコンテナーのエラー

問題

TALM のマネージャーコンテナーのログを確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

出力例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error
{"reconciler group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade",
"name": "lab-upgrade", "namespace": "default", "error": "Cluster spoke5555 is not a
ManagedCluster"} 1
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

1

エラーを表示します。

ClusterGroupUpgrade CR が完了した後、クラスターが一部のポリシーに準拠していない

問題

修復が必要かどうかを判断するために TALM が使用するポリシーコンプライアンスステータスは、まだすべてのクラスターで完全に更新されていません。これには次の理由が考えられます。

- ポリシーの作成または更新後、CGU の実行が早すぎました。

- ポリシーの修復は、ClusterGroupUpgrade CR の後続のポリシーのコンプライアンスに影響します。

解決方法

同じ仕様で新しい ClusterGroupUpdate CR を作成して適用します。

GitOps ZTP ワークフローで自動作成された ClusterGroupUpgrade CR に管理ポリシーがない

問題

クラスターが Ready になったときにマネージドクラスターのポリシーがない場合、ポリシーのない ClusterGroupUpgrade CR が自動作成されます。ClusterGroupUpgrade CR が完了すると、マネージドクラスターには ztp-done というラベルが付けられます。SiteConfig リソースがプッシュされた後、必要な時間内に PolicyGenTemplate CR が Git リポジトリーにプッシュされなかった場合、クラスターが Ready になったときに、ターゲットクラスターで使用できるポリシーがなくなる可能性があります。

解決方法

適用するポリシーがハブクラスターで使用可能であることを確認してから、必要なポリシーを使用して ClusterGroupUpgrade CR を作成します。

ClusterGroupUpgrade CR を手動で作成するか、自動作成を再度トリガーすることができます。ClusterGroupUpgrade CR の自動作成をトリガーするには、クラスターから ztp-done ラベルを削除し、以前に zip-install 名前空間で作成された空の ClusterGroupUpgrade CR を削除します。

事前キャッシュに失敗しました

問題

事前キャッシュは、次のいずれかの理由で失敗する場合があります。

- ノードに十分な空き容量がありません。

- 切断された環境では、事前キャッシュイメージが適切にミラーリングされていません。

- Pod の作成中に問題が発生しました。

解決方法

1. スペース不足のために事前キャッシュが失敗したかどうかを確認するには、ノードの事前キャッシュ Pod のログを確認します。

   a. 次のコマンドを使用して Pod の名前を見つけます。

   ```
   $ oc get pods -n openshift-talo-pre-cache
   ```

   b. 次のコマンドを使用してログをチェックし、エラーが容量不足に関連しているかどうかを確認します。

   ```
   $ oc logs -n openshift-talo-pre-cache <pod name>
   ```

2. ログがない場合は、次のコマンドを使用して Pod のステータスを確認します。

   ```
   $ oc describe pod -n openshift-talo-pre-cache <pod name>
   ```

3. Pod が存在しない場合は、次のコマンドを使用してジョブのステータスをチェックし、Pod を作成できなかった理由を確認します。

   ```
   $ oc describe job -n openshift-talo-pre-cache pre-cache
   ```

関連情報

- OpenShift Container Platform Troubleshooting Operator の問題
- Topology Aware Lifecycle Manager を使用した管理ポリシーの更新
- PolicyGenerator CRD について

第13章 GITOPS ZTP を使用した単一ノードの OPENSHIFT クラスターの拡張

GitOps Zero Touch Provisioning (ZTP) を使用して、シングルノード OpenShift クラスターを拡張できます。単一ノードの OpenShift クラスターにワーカーノードを追加すると、元の単一ノードの OpenShift クラスターがコントロールプレーンノードのロールを保持します。ワーカーノードを追加しても、既存の単一ノード OpenShift クラスターのダウンタイムは必要ありません。

注記

単一ノードの OpenShift クラスターに追加できるワーカーノードの数に指定された制限はありませんが、追加のワーカーノード用にコントロールプレーンノードで予約されている CPU 割り当てを再評価する必要があります。

ワーカーノードでワークロードパーティショニングが必要な場合は、ノードをインストールする前に、ハブクラスターでマネージドクラスターポリシーをデプロイして修正する必要があります。そうすることで、GitOps ZTP ワークフローが MachineConfig Ignition ファイルをワーカーノードに適用する前に、ワークロードパーティショニング MachineConfig オブジェクトがレンダリングされ、worker マシン設定プールに関連付けられます。

最初にポリシーを修正してから、ワーカーノードをインストールすることを推奨します。ワーカーノードのインストール後にワークロードパーティショニングマニフェストを作成する場合は、ノードを手動でドレインし、デーモンセットによって管理されるすべての Pod を削除する必要があります。管理デーモンセットが新しい Pod を作成すると、新しい Pod はワークロードパーティショニングプロセスを実行します。

重要

GitOps ZTP を使用した単一ノードの OpenShift クラスターへのワーカーノードの追加は、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 製品サポート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場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

関連情報

- vDU アプリケーションのデプロイ用に調整された単一ノードの OpenShift クラスターの詳

細は、単一ノードの OpenShift に vDU をデプロイするためのリファレンス設定 を参照してください。

- ワーカーノードの詳細は、シングルノード OpenShift クラスターへのワーカーノードの追加 を参照してください。

- 拡張シングルノード OpenShift クラスターからワーカーノードを削除する方法については、コマンドラインインターフェイスを使用してマネージドクラスターノードを削除する を参照してください。

13.1. POLICYGENERATOR または POLICYGENTEMPLATE リソースを使用したワーカーノードへのプロファイルの適用

DU プロファイルを使用して、追加のワーカーノードを設定できます。

GitOps Zero Touch Provisioning (ZTP) 共通、グループ、およびサイト固有の PolicyGenTemplate リソースを使用して、RAN 分散ユニット (DU) プロファイルをワーカーノードクラスターに適用できます。ArgoCD policies アプリケーションにリンクされている GitOps ZTP パイプラインには、ztp-site-generate コンテナーを抽出するときに out/argocd/example/policygentemplates フォルダーにある次の CR が含まれています。

/acmpolicygenerator resources

- acm-common-ranGen.yaml
- acm-group-du-sno-ranGen.yaml
- acm-example-sno-site.yaml
- ns.yaml
- kustomization.yaml

/policygentemplates リソース

- common-ranGen.yaml
- group-du-sno-ranGen.yaml
- example-sno-site.yaml
- ns.yaml
- kustomization.yaml

ワーカーノードでの DU プロファイルの設定は、アップグレードと見なされます。アップグレードフローを開始するには、既存のポリシーを更新するか、追加のポリシーを作成する必要があります。次に、ClusterGroupUpgrade CR を作成して、クラスターのグループ内のポリシーを調整する必要があります。

13.2. (オプション) PTP および SR-IOV デーモンセレクターの互換性の確保

DU プロファイルが GitOps Zero Touch Provisioning (ZTP) プラグインバージョン 4.11 以前を使用してデプロイされた場合、PTP および SR-IOV Operator は、master というラベルの付いたノードにのみデーモンを配置するように設定されている可能性があります。この設定により、PTP および SR-IOV デーモンがワーカーノードで動作しなくなります。システムで PTP および SR-IOV デーモンノードセレクターが正しく設定されていない場合は、ワーカー DU プロファイル設定に進む前にデーモンを変更する必要があります。

手順

1. スポーククラスターの 1 つで PTP Operator のデーモンノードセレクター設定を確認します。

```
$ oc get ptpoperatorconfig/default -n openshift-ptp -ojsonpath='{.spec}' | jq
```

PTP Operator の出力例

```
{"daemonNodeSelector":{"node-role.kubernetes.io/master":""}} (1)
```

1

ノードセレクターが master に設定されている場合、スポークは、変更が必要なバージョンの GitOps ZTP プラグインでデプロイされています。

2. スポーククラスターの 1 つで SR-IOV Operator のデーモンノードセレクター設定を確認します。

```
$  oc get sriovoperatorconfig/default -n \
openshift-sriov-network-operator -ojsonpath='{.spec}' | jq
```

SR-IOV Operator の出力例

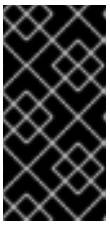
```
{"configDaemonNodeSelector":{"node-
role.kubernetes.io/worker":""},"disableDrain":false,"enableInjector":true,"enableOpera
torWebhook":true} 1
```

1

ノードセレクターが master に設定されている場合、スポークは、変更が必要なバージョンの GitOps ZTP プラグインでデプロイされています。

3. グループポリシーで、次の ComplianceType および spec エントリーを追加します。

```
spec:
  - fileName: PtpOperatorConfig.yaml
    policyName: "config-policy"
    complianceType: mustonlyhave
    spec:
      daemonNodeSelector:
        node-role.kubernetes.io/worker: ""
  - fileName: SriovOperatorConfig.yaml
    policyName: "config-policy"
    complianceType: mustonlyhave
    spec:
      configDaemonNodeSelector:
        node-role.kubernetes.io/worker: ""
```



重要

daemonNodeSelector フィールドを変更すると、一時的な PTP Synchronization が失われ、SR-IOV 接続が失われます。

4. Git で変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視されている Git リポジトリーにプッシュします。

13.3. PTP および SR-IOV ノードセレクターの互換性

PTP 設定リソースと SR-IOV ネットワークノードポリシーは、ノードセレクターとして node-role.kubernetes.io/master: "" を使用します。追加のワーカーノードの NIC 設定がコントロールプレーンノードと同じである場合、コントロールプレーンノードの設定に使用されたポリシーをワーカーノードに再利用できます。ただし、両方のノードタイプを選択するようにノードセレクターを変更する必要があります (たとえば、node-role.kubernetes.io/worker ラベルを使用)。

13.4. POLICYGENERATOR CR を使用してワーカーノードポリシーをワーカーノードに適用する

PolicyGenerator CR を使用して、ワーカーノードのポリシーを作成できます。

手順

1. 以下の PolicyGenerator CR を作成します。

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: example-sno-workers
placementBindingDefaults:
  name: example-sno-workers-placement-binding
policyDefaults:
  namespace: example-sno
  placement:
    labelSelector:
      matchExpressions:
        - key: sites
          operator: In
          values:
            - example-sno 1
  remediationAction: inform
  severity: low
  namespaceSelector:
    exclude:
      - kube-*
```

```
      include:
        - '*'
    evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
  - name: example-sno-workers-config-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "10"
    manifests:
      - path: source-crs/MachineConfigGeneric.yaml 2
        patches:
          - metadata:
              labels:
                machineconfiguration.openshift.io/role: worker 3
              name: enable-workload-partitioning
            spec:
              config:
                storage:
                  files:
                    - contents:
                        source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhd
GlvbI9hbm5vdGF0aW9uID0glnRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFu
YWdlbWVudCIKYW5ub3RhdGlvbI9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm
9wZW5zaGlmdC5pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHU
D0gIjAtMyIgfQo=
                      mode: 420
                      overwrite: true
                      path: /etc/crio/crio.conf.d/01-workload-partitioning
                      user:
                        name: root
                    - contents:
                        source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg
==
                      mode: 420
                      overwrite: true
                      path: /etc/kubernetes/openshift-workload-pinning
                      user:
                        name: root
      - path: source-crs/PerformanceProfile-MCP-worker.yaml
        patches:
          - metadata:
              name: openshift-worker-node-performance-profile
            spec:
              cpu: 4
                isolated: 4-47
                reserved: 0-3
              hugepages:
                defaultHugepagesSize: 1G
                pages:
                  - count: 32
                    size: 1G
              realTimeKernel:
                enabled: true
```

```
    - path: source-crs/TunedPerformancePatch-MCP-worker.yaml
      patches:
        - metadata:
            name: performance-patch-worker
          spec:
            profile:
              - data: |
                [main]
                summary=Configuration changes profile inherited from performance
created tuned
                include=openshift-node-performance-openshift-worker-node-
performance-profile
                [bootloader]
                cmdline_crash=nohz_full=4-47 5
                [sysctl]
                kernel.timer_migration=1
                [scheduler]
                group.ice-ptp=0:f:10:*:ice-ptp.*
                [service]
                service.stalld=start,enable
                service.chronyd=stop,disable
                name: performance-patch-worker
            recommend:
              - profile: performance-patch-worker
```

1

ポリシーは、このラベルを持つすべてのクラスターに適用されます。

2

この汎用の MachineConfig CR は、ワーカーノードでワークロードの分割を設定するために使用されます。

3

MCP フィールドは worker に設定する必要があります。

4

cpu.isolated および cpu.reserved フィールドは、特定のハードウェアプラットフォームごとに設定する必要があります。

5

cmdline_crash CPU セットは、PerformanceProfile セクションの cpu.isolated セットと一致する必要があります。

汎用の MachineConfig CR を使用して、ワーカーノードでワークロードパーティションを

設定します。crio および kubelet 設定ファイルのコンテンツを生成できます。

2. 作成したポリシーテンプレートを、ArgoCD policies アプリケーションによってモニターされている Git リポジトリーに追加します。

3. ポリシーを kustomization.yaml ファイルに追加します。

4. Git で変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視されている Git リポジトリーにプッシュします。

5. 新しいポリシーをスポーククラスターに修正するには、TALM カスタムリソースを作成します。

```
$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
  - example-sno
  enable: true
  managedPolicies:
  - group-du-sno-config-policy
  - example-sno-workers-config-policy
  - example-sno-config-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
EOF
```

13.5. POLICYGENTEMPLATE CR を使用してワーカーノードポリシーをワーカーノードに適用する

PolicyGenTemplate CR を使用してワーカーノードのポリシーを作成できます。

手順

1. 次の PolicyGenTemplate CR を作成します。

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

```
kind: PolicyGenTemplate
metadata:
  name: "example-sno-workers"
  namespace: "example-sno"
spec:
  bindingRules:
    sites: "example-sno" (1)
  mcp: "worker" (2)
  sourceFiles:
    - fileName: MachineConfigGeneric.yaml (3)
      policyName: "config-policy"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: worker
        name: enable-workload-partitioning
      spec:
        config:
          storage:
            files:
            - contents:
                source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhd
Glvbl9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFu
YWdlbWVudCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm
9wZW5zaGlmdC5pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHU
D0gIjAtMyIgfQo=
              mode: 420
              overwrite: true
              path: /etc/crio/crio.conf.d/01-workload-partitioning
              user:
                name: root
            - contents:
                source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg
==
              mode: 420
              overwrite: true
              path: /etc/kubernetes/openshift-workload-pinning
              user:
                name: root
    - fileName: PerformanceProfile.yaml
      policyName: "config-policy"
      metadata:
        name: openshift-worker-node-performance-profile
      spec:
        cpu: (4)
          isolated: "4-47"
          reserved: "0-3"
        hugepages:
          defaultHugepagesSize: 1G
          pages:
            - size: 1G
              count: 32
        realTimeKernel:
          enabled: true
    - fileName: TunedPerformancePatch.yaml
```

```
    policyName: "config-policy"
    metadata:
      name: performance-patch-worker
    spec:
      profile:
        - name: performance-patch-worker
          data: |
            [main]
            summary=Configuration changes profile inherited from performance created
tuned
            include=openshift-node-performance-openshift-worker-node-performance-
profile
            [bootloader]
            cmdline_crash=nohz_full=4-47 5
            [sysctl]
            kernel.timer_migration=1
            [scheduler]
            group.ice-ptp=0:f:10:*:ice-ptp.*
            [service]
            service.stalld=start,enable
            service.chronyd=stop,disable
      recommend:
      - profile: performance-patch-worker
```

1

ポリシーは、このラベルを持つすべてのクラスターに適用されます。

2

MCP フィールドは worker に設定する必要があります。

3

この汎用の MachineConfig CR は、ワーカーノードでワークロードの分割を設定するために使用されます。

4

cpu.isolated および cpu.reserved フィールドは、特定のハードウェアプラットフォームごとに設定する必要があります。

5

cmdline_crash CPU セットは、PerformanceProfile セクションの cpu.isolated セットと一致する必要があります。

汎用の MachineConfig CR を使用して、ワーカーノードでワークロードパーティションを設定します。crio および kubelet 設定ファイルのコンテンツを生成できます。

2. 作成したポリシーテンプレートを、ArgoCD policies アプリケーションによってモニターされている Git リポジトリーに追加します。

3. ポリシーを kustomization.yaml ファイルに追加します。

4. Git で変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視されている Git リポジトリーにプッシュします。

5. 新しいポリシーをスポーククラスターに修正するには、TALM カスタムリソースを作成します。

```
$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
  - example-sno
  enable: true
  managedPolicies:
  - group-du-sno-config-policy
  - example-sno-workers-config-policy
  - example-sno-config-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
EOF
```

13.6. GITOPS ZTP を使用して単一ノードの OPENSHIFT クラスターにワーカーノードを追加する

1 つ以上のワーカーノードを既存の単一ノード OpenShift クラスターに追加して、クラスターで使用可能な CPU リソースを増やすことができます。

前提条件

- OpenShift Container Platform 4.11 以降のベアメタルハブクラスターに RHACM 2.6 以降をインストールして設定する

- ハブクラスターに Topology Aware Lifecycle Manager をインストールする
- ハブクラスターに Red Hat OpenShift GitOps をインストールする
- GitOps ZTP ztp-site-generate コンテナーイメージバージョン 4.12 以降を使用する
- GitOps ZTP を使用して管理対象の単一ノード OpenShift クラスターをデプロイする
- RHACM ドキュメントの説明に従って、中央インフラストラクチャー管理を設定する
- 内部 API エンドポイント api-int.<cluster_name>.<base_domain> を解決するようにクラスターにサービスを提供する DNS を設定する

手順

1. example-sno.yaml SiteConfig マニフェストを使用してクラスターをデプロイした場合は、新しいワーカーノードを spec.clusters['example-sno'].nodes リストに追加します。

```
nodes:
- hostName: "example-node2.example.com"
  role: "worker"
  bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embed
ded.1"
  bmcCredentialsName:
    name: "example-node2-bmh-secret"
  bootMACAddress: "AA:BB:CC:DD:EE:11"
  bootMode: "UEFI"
  nodeNetwork:
    interfaces:
      - name: eno1
        macAddress: "AA:BB:CC:DD:EE:11"
    config:
      interfaces:
        - name: eno1
          type: ethernet
          state: up
          macAddress: "AA:BB:CC:DD:EE:11"
          ipv4:
            enabled: false
          ipv6:
            enabled: true
```

```
      address:
      - ip: 1111:2222:3333:4444::1
        prefix-length: 64
  dns-resolver:
    config:
      search:
      - example.com
      server:
      - 1111:2222:3333:4444::2
  routes:
    config:
    - destination: ::/0
      next-hop-interface: eno1
      next-hop-address: 1111:2222:3333:4444::1
      table-id: 254
```

2. SiteConfig ファイルの spec.nodes セクションの bmcCredentialsName フィールドで参照されるように、新しいホストの BMC 認証シークレットを作成します。

```
apiVersion: v1
data:
  password: "password"
  username: "username"
kind: Secret
metadata:
  name: "example-node2-bmh-secret"
  namespace: example-sno
type: Opaque
```

3. Git で変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視されている Git リポジトリーにプッシュします。

ArgoCD cluster アプリケーションが同期すると、GitOps ZTP プラグインによって生成されたハブクラスターに 2 つの新しいマニフェストが表示されます。

- BareMetalHost
- NMStateConfig

重要

cpuset フィールドは、ワーカーノードに対して設定しないでください。ワーカーノードのワークロードパーティショニングは、ノードのインストールが完了した後、管理ポリシーを通じて追加されます。

検証

インストールプロセスは、いくつかの方法でモニターできます。

- 次のコマンドを実行して、事前プロビジョニングイメージが作成されているかどうかを確認します。

```
$ oc get ppimg -n example-sno
```

出力例

```
NAMESPACE      NAME           READY   REASON
example-sno    example-sno    True    ImageCreated
example-sno    example-node2  True    ImageCreated
```

- ベアメタルホストの状態を確認します。

```
$ oc get bmh -n example-sno
```

出力例

```
NAME           STATE         CONSUMER   ONLINE   ERROR   AGE
example-sno    provisioned              true             69m
example-node2  provisioning             true             4m50s 1
```

1 provisioning ステータスは、インストールメディアからのノードの起動が進行中であることを示します。

- インストールプロセスを継続的に監視します。

a. 次のコマンドを実行して、エージェントのインストールプロセスを監視します。

```
$ oc get agent -n example-sno --watch
```

出力例

```
NAME                                   CLUSTER   APPROVED   ROLE     STAGE
671bc05d-5358-8940-ec12-d9ad22804faa   example-sno   true       master   Done
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Starting
installation
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Installing
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Writing
image to disk
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Waiting
for control plane
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Rebooting
14fd821b-a35d-9cba-7978-00ddf535ff37   example-sno   true       worker   Done
```

b. ワーカーノードのインストールが完了すると、ワーカーノードの証明書が自動的に承認されます。この時点で、ワーカーは ManagedClusterInfo ステータスで表示されます。次のコマンドを実行して、ステータスを確認します。

```
$ oc get managedclusterinfo/example-sno -n example-sno -o \
jsonpath='{range .status.nodeList[*]}{.name}{"\t"}{.conditions}{"\t"}{.labels}{"\n"}
{end}'
```

出力例

```
example-sno [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/master":"","node-role.kubernetes.io/worker":""}
example-node2 [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/worker":""}
```

第14章 単一ノードの OPENSHIFT デプロイメント用のイメージの事前キャッシュ

GitOps Zero Touch Provisioning (ZTP) ソリューションを使用して多数のクラスターをデプロイする、帯域幅が制限された環境では、OpenShift Container Platform のブートストラップとインストールに必要なすべてのイメージをダウンロードすることを避ける必要があります。リモートの単一ノードの OpenShift サイトでは帯域幅が制限されているため、デプロイに時間がかかる場合があります。factory-precaching-cli ツールを使用すると、ZTP プロビジョニングのためにサーバーをリモートサイトに出荷する前にサーバーを事前にステージングできます。

factory-precaching-cli ツールは次のことを行います。

- 最小限の ISO の起動に必要な RHCOS rootfs イメージをダウンロードします。
- data というラベルの付いたインストールディスクからパーティションを作成します。
- ディスクを xfs でフォーマットします。
- ディスクの最後に GUID パーティションテーブル (GPT) データパーティションを作成します。パーティションのサイズはツールで設定できます。
- OpenShift Container Platform のインストールに必要なコンテナーイメージをコピーします。
- OpenShift Container Platform をインストールするために ZTP が必要とするコンテナーイメージをコピーします。
- オプション: Day-2 Operator をパーティションにコピーします。

重要

factory-precaching-cli ツールは、テクノロジープレビュー機能専用です。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

14.1. FACTORY-PRECACHING-CLI ツールの入手

factory-precaching-cli ツールの Go バイナリーは、{rds-first} tools container image で公開されています。コンテナーイメージ内の factory-precaching-cli ツール Go バイナリーは、podman を使用して RHCOS ライブイメージを実行しているサーバー上で実行されます。切断された環境で作業している場合、またはプライベートレジストリーがある場合は、そこにイメージをコピーして、イメージをサーバーにダウンロードできるようにする必要があります。

手順

- 次のコマンドを実行して、factory-precaching-cli ツールイメージをプルします。

```
# podman pull quay.io/openshift-kni/telco-ran-tools:latest
```

検証

- ツールが利用可能であることを確認するには、factory-precaching-cli ツール Go バイナリーの現在のバージョンを照会します。

```
# podman run quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli -v
```

出力例

```
factory-precaching-cli version 20221018.120852+main.feecf17
```

14.2. ライブオペレーティングシステムイメージからの起動

factory-precaching-cli ツールを使用して、1 つのディスクしか使用できず、外部ディスクドライブをサーバーに接続できないサーバーを起動できます。

警告

RHCOS では、ディスクが RHCOS イメージで書き込まれようとしているときに、ディスクが使用されていない必要があります。

サーバーハードウェアに応じて、次のいずれかの方法を使用して、空のサーバーに RHCOS ライブ ISO をマウントできます。

- Dell サーバーで Dell RACADM ツールを使用する。
- HP サーバーで HPONCFG ツールを使用する。
- Redfish BMC API を使用する。

注記

マウント手順を自動化することを推奨します。手順を自動化するには、必要なイメージをプルして、ローカル HTTP サーバーでホストする必要があります。

前提条件

- ホストの電源を入れた。
- ホストへのネットワーク接続がある。

手順

この例の手順では、Redfish BMC API を使用して RHCOS ライブ ISO をマウントします。

1. RHCOS ライブ ISO をマウントします。

   a. 仮想メディアのステータスを確認します。

   ```
   $ curl --globoff -H "Content-Type: application/json" -H \
   "Accept: application/json" -k -X GET --user ${username_password} \
   https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1 | python -m
   json.tool
   ```

   b. ISO ファイルを仮想メディアとしてマウントします。

   ```
   $ curl --globoff -L -w "%{http_code} %{url_effective}\\n" -ku
   ${username_password} -H "Content-Type: application/json" -H "Accept:
   application/json" -d '{"Image": "http://[$HTTPd_IP]/RHCOS-live.iso"}' -X POST
   https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1/Actions/VirtualMe
   dia.InsertMedia
   ```

   c. 仮想メディアから 1 回起動するように起動順序を設定します。

   ```
   $ curl --globoff  -L -w "%{http_code} %{url_effective}\\n"  -ku
   ${username_password}  -H "Content-Type: application/json" -H "Accept:
   application/json" -d '{"Boot":{ "BootSourceOverrideEnabled": "Once",
   "BootSourceOverrideTarget": "Cd", "BootSourceOverrideMode": "UEFI"}}' -X
   PATCH https://$BMC_ADDRESS/redfish/v1/Systems/Self
   ```

2. 再起動し、サーバーが仮想メディアから起動していることを確認します。

関連情報

- butane ユーティリティーの詳細は、Butane について を参照してください。
- カスタムライブ RHCOS ISO の作成の詳細は、リモートサーバーアクセス用のカスタムライブ RHCOS ISO の作成 を参照してください。

- Dell RACADM ツールの使用の詳細については、Integrated Dell Remote Access Controller 9 RACADM CLI Guide を参照してください。

- HP HPONCFG ツールの使用の詳細については、HPONCFG の 使用 を参照してください。

- Redfish BMC API の使用に関する詳細は、Redfish API を使用した HTTP ホスト ISO イメージからの起動 を参照してください。

14.3. ディスクのパーティション設定

完全な事前キャッシュプロセスを実行するには、ライブ ISO から起動し、コンテナーイメージから factory-precaching-cli ツールを使用して、必要なすべてのアーティファクトを分割および事前キャッシュする必要があります。

プロビジョニング中にオペレーティングシステム (RHCOS) がデバイスに書き込まれるときにディスクが使用されていてはならないため、ライブ ISO または RHCOS ライブ ISO が必要です。この手順で単一ディスクサーバーを有効にすることもできます。

前提条件

- パーティショニングされていないディスクがある。

- quay.io/openshift-kni/telco-ran-tools:latest イメージにアクセスできます。

- OpenShift Container Platform をインストールし、必要なイメージを事前キャッシュするのに十分なストレージがある。

手順

1. ディスクがクリアされていることを確認します。

```
# lsblk
```

出力例

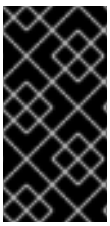
```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0     7:0    0  93.8G  0 loop /run/ephemeral
loop1     7:1    0 897.3M  1 loop /sysroot
sr0      11:0    1   999M  0 rom  /run/media/iso
nvme0n1 259:1    0   1.5T  0 disk
```

2. ファイルシステム、RAID、またはパーティションテーブルの署名をデバイスから消去します。

```
# wipefs -a /dev/nvme0n1
```

出力例

```
/dev/nvme0n1: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 54
/dev/nvme0n1: 8 bytes were erased at offset 0x1749a955e00 (gpt): 45 46 49 20 50 41 52
54
/dev/nvme0n1: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
```



重要

ディスクが空でない場合、ツールはデバイスのパーティション番号 1 を使用してアーティファクトを事前キャッシュするため、失敗します。

14.3.1. パーティションの作成

デバイスの準備ができたら、単一のパーティションと GPT パーティションテーブルを作成します。パーティションは自動的に data としてラベル付けされ、デバイスの最後に作成されます。そうしないと、パーティションは coreos-installer によって上書きされます。

重要

coreos-installer では、パーティションをデバイスの最後に作成し、data としてラベル付けする必要があります。RHCOS イメージをディスクに書き込むときにパーティションを保存するには、両方の要件が必要です。

前提条件

- ホストデバイスがフォーマットされているため、コンテナーは privileged として実行する必要があります。
- コンテナー内でプロセスを実行できるように、/dev フォルダーをマウントする必要があります。

手順

次の例では、Day 2 Operator の DU プロファイルを事前キャッシュできるようにするため、パーティションのサイズは 250 GiB です。

1. コンテナーを privileged として実行し、ディスクをパーティショニングします。

```
# podman run -v /dev:/dev --privileged \
--rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli partition \ 1
-d /dev/nvme0n1 \ 2
-s 250 3
```

1. factory-precaching-cli ツールのパーティショニング機能を指定します。
2. ディスク上のルートディレクトリーを定義します。
3. ディスクのサイズを GB 単位で定義します。

2. ストレージ情報を確認します。

```
# lsblk
```

出力例

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:3    0   250G  0 part
```

検証

次の要件が満たされていることを確認する必要があります。

- デバイスには GPT パーティションテーブルがあります。
- パーティションは、デバイスの最新のセクターを使用します。
- パーティションは data として正しくラベル付けされています。

ディスクのステータスを照会して、ディスクが期待どおりにパーティショニングされていることを確認します。

```
# gdisk -l /dev/nvme0n1
```

出力例

```
GPT fdisk (gdisk) version 1.0.3

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
Disk /dev/nvme0n1: 3125627568 sectors, 1.5 TiB
Model: Dell Express Flash PM1725b 1.6TB SFF
Sector size (logical/physical): 512/512 bytes
Disk identifier (GUID): CB5A9D44-9B3C-4174-A5C1-C64957910B61
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
```

```
First usable sector is 34, last usable sector is 3125627534
Partitions will be aligned on 2048-sector boundaries
Total free space is 2601338846 sectors (1.2 TiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1      2601338880      3125627534   250.0 GiB   8300  data
```

14.3.2. パーティションのマウント

ディスクが正しくパーティショニングされていることを確認したら、デバイスを /mnt にマウントできます。

重要

GitOps ZTP の準備中にそのマウントポイントが使用されるため、デバイスを /mnt にマウントすることを推奨します。

1. パーティションが xfs としてフォーマットされていることを確認します。

```
# lsblk -f /dev/nvme0n1
```

出力例

```
NAME        FSTYPE LABEL UUID                                 MOUNTPOINT
nvme0n1
└─nvme0n1p1 xfs          1bee8ea4-d6cf-4339-b690-a76594794071
```

2. パーティションをマウントします。

```
# mount /dev/nvme0n1p1 /mnt/
```

検証

- パーティションがマウントされていることを確認します。

```
# lsblk
```

出力例

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  93.8G  0 loop /run/ephemeral
loop1         7:1    0 897.3M  1 loop /sysroot
sr0          11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:2    0   250G  0 part /var/mnt (1)
```

(1) RHCOS の /mnt フォルダーは /var/mnt へのリンクであるため、マウントポイントは /var/mnt です。

14.4. イメージのダウンロード

factory-precaching-cli ツールを使用すると、パーティショニングされたサーバーに次のイメージをダウンロードできます。

- OpenShift Container Platform イメージ
- 5G RAN サイトの分散ユニット (DU) プロファイルに含まれる Operator イメージ
- 切断されたレジストリーからの Operator イメージ

注記

使用可能な Operator イメージのリストは、OpenShift Container Platform リリースごとに異なる場合があります。

14.4.1. 並列ワーカーを使用したダウンロード

factory-precaching-cli ツールは、並列ワーカーを使用して複数のイメージを同時にダウンロードします。--parallel または -p オプションを使用して、ワーカーの数を設定できます。デフォルトの数値は、サーバーで使用可能な CPU の 80% に設定されています。

注記

ログインシェルが CPU のサブセットに制限されている可能性があります。その場合、コンテナーで使用できる CPU が減少します。この制限を取り除くには、コマンドの前に taskset 0xffffffff を付けます。次に例を示します。

```
# taskset 0xffffffff podman run --rm quay.io/openshift-kni/telco-ran-tools:latest
factory-precaching-cli download --help
```

14.4.2. OpenShift Container Platform イメージのダウンロードの準備

OpenShift Container Platform コンテナーイメージをダウンロードするには、マルチクラスターエンジンのバージョンを知る必要があります。--du-profile フラグを使用する場合は、単一ノードの OpenShift をプロビジョニングするハブクラスターで実行されている Red Hat Advanced Cluster Management (RHACM) のバージョンも指定する必要があります。

前提条件

- RHACM とマルチクラスターエンジン Operator がインストールされている。
- ストレージデバイスをパーティショニングしている。
- パーティショニングされたデバイスにイメージ用の十分なスペースがある。
- ベアメタルサーバーをインターネットに接続している。
- 有効なプルシークレットがあります。

手順

1. ハブクラスターで次のコマンドを実行して、RHACM バージョンとマルチクラスターエン

ジンバージョンを確認します。

```
$ oc get csv -A | grep -i advanced-cluster-management
```

出力例

```
open-cluster-management                advanced-cluster-management.v2.6.3
Advanced Cluster Management for Kubernetes   2.6.3                advanced-cluster-
management.v2.6.3                Succeeded
```

```
$ oc get csv -A | grep -i multicluster-engine
```

出力例

```
multicluster-engine                    cluster-group-upgrades-operator.v0.0.3
cluster-group-upgrades-operator          0.0.3
Pending
multicluster-engine                    multicluster-engine.v2.1.4                multicluster
engine for Kubernetes          2.1.4              multicluster-engine.v2.0.3
Succeeded
multicluster-engine                    openshift-gitops-operator.v1.5.7          Red Hat
OpenShift GitOps               1.5.7              openshift-gitops-operator.v1.5.6-
0.1664915551.p   Succeeded
multicluster-engine                    openshift-pipelines-operator-rh.v1.6.4    Red
Hat OpenShift Pipelines        1.6.4              openshift-pipelines-operator-rh.v1.6.3
Succeeded
```

2. コンテナーレジストリーにアクセスするには、インストールするサーバーに有効なプルシークレットをコピーします。

   a. .docker フォルダーを作成します。

   ```
   $ mkdir /root/.docker
   ```

   b. config.json ファイルの有効なプルを、以前に作成した .docker/ フォルダーにコピー

します。

```
$ cp config.json /root/.docker/config.json 1
```

1

/root/.docker/config.json は、podman がレジストリーのログイン認証情報をチェックするデフォルトのパスです。

注記

別のレジストリーを使用して必要なアーティファクトをプルする場合は、適切なプルシークレットをコピーする必要があります。ローカルレジストリーが TLS を使用している場合は、レジストリーからの証明書も含める必要があります。

14.4.3. OpenShift Container Platform イメージのダウンロード

factory-precaching-cli ツールを使用すると、特定の OpenShift Container Platform リリースをプロビジョニングするために必要なすべてのコンテナーイメージを事前キャッシュできます。

手順

- 次のコマンドを実行して、リリースを事前キャッシュします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools -- \
  factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
```

1

factory-precaching-cli ツールのダウンロード機能を指定します。

2

OpenShift Container Platform リリースバージョンを定義します。

3

RHACM バージョンを定義します。

4

マルチクラスターエンジンのバージョンを定義します。

5

ディスク上のイメージをダウンロードするフォルダーを定義します。

6

オプション: 追加のイメージを保存するリポジトリーを定義します。これらのイメージはダウンロードされ、ディスクに事前キャッシュされます。

出力例

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
Processing artifact [1/176]: ocp-v4.0-art-dev@sha256_6ac2b96bf4899c01a87366fd0feae9f57b1b61878e3b5823da0c3f34f707fbf5
Processing artifact [2/176]: ocp-v4.0-art-dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657c
Processing artifact [3/176]: ocp-v4.0-art-dev@sha256_a480390e91b1c07e10091c3da2257180654f6b2a735a4ad4c3b69dbdb77bbc06
Processing artifact [4/176]: ocp-v4.0-art-dev@sha256_ecc5d8dbd77e326dba6594ff8c2d091eefbc4d90c963a9a85b0b2f0e6155f995
Processing artifact [5/176]: ocp-v4.0-art-dev@sha256_274b6d561558a2f54db08ea96df9892315bb773fc203b1dbcea418d20f4c7ad1
Processing artifact [6/176]: ocp-v4.0-art-dev@sha256_e142bf5020f5ca0d1bdda0026bf97f89b72d21a97c9cc2dc71bf85050e822bbf
...
Processing artifact [175/176]: ocp-v4.0-art-dev@sha256_16cd7eda26f0fb0fc965a589e1e96ff8577e560fcd14f06b5fda1643036ed6c8
Processing artifact [176/176]: ocp-v4.0-art-dev@sha256_cf4d862b4a4170d4f611b39d06c31c97658e309724f9788e155999ae51e7188f
...
Summary:

Release:                4.16.0
Hub Version:            2.6.3
ACM Version:            2.6.3
```

```
MCE Version:                    2.1.4
Include DU Profile:             No
Workers:                        83
```

検証

- すべてのイメージがサーバーのターゲットフォルダーに圧縮されていることを確認します。

```
$ ls -l /mnt 1
```

1

/mnt フォルダーにイメージを事前キャッシュしておくことを推奨します。

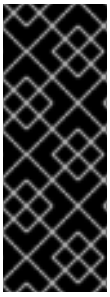
出力例

```
-rw-r--r--. 1 root root  136352323 Oct 31 15:19 ocp-v4.0-art-
dev@sha256_edec37e7cd8b1611d0031d45e7958361c65e2005f145b471a8108f1b54316c
07.tgz
-rw-r--r--. 1 root root  156092894 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_ee51b062b9c3c9f4fe77bd5b3cc9a3b12355d040119a1434425a824f137c61a
9.tgz
-rw-r--r--. 1 root root  172297800 Oct 31 15:29 ocp-v4.0-art-
dev@sha256_ef23d9057c367a36e4a5c4877d23ee097a731e1186ed28a26c8d21501cd827
18.tgz
-rw-r--r--. 1 root root  171539614 Oct 31 15:23 ocp-v4.0-art-
dev@sha256_f0497bb63ef6834a619d4208be9da459510df697596b891c0c633da144dbb0
25.tgz
-rw-r--r--. 1 root root  160399150 Oct 31 15:20 ocp-v4.0-art-
dev@sha256_f0c339da117cde44c9aae8d0bd054bceb6f19fdb191928f6912a703182330ac
2.tgz
-rw-r--r--. 1 root root  175962005 Oct 31 15:17 ocp-v4.0-art-
dev@sha256_f19dd2e80fb41ef31d62bb8c08b339c50d193fdb10fc39cc15b353cbbfeb9b2
4.tgz
-rw-r--r--. 1 root root  174942008 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_f1dbb81fa1aa724e96dd2b296b855ff52a565fbef003d08030d63590ae6454d
f.tgz
-rw-r--r--. 1 root root  246693315 Oct 31 15:31 ocp-v4.0-art-
dev@sha256_f44dcf2c94e4fd843cbbf9b11128df2ba856cd813786e42e3da1fdfb0f6ddd01
.tgz
-rw-r--r--. 1 root root  170148293 Oct 31 15:00 ocp-v4.0-art-
dev@sha256_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657
c.tgz
-rw-r--r--. 1 root root  168899617 Oct 31 15:16 ocp-v4.0-art-
```

```
dev@sha256_f5099b0989120a8d08a963601214b5c5cb23417a707a8624b7eb52ab788a7f
75.tgz
-rw-r--r--. 1 root root  176592362 Oct 31 15:05 ocp-v4.0-art-
dev@sha256_f68c0e6f5e17b0b0f7ab2d4c39559ea89f900751e64b97cb42311a478338d9c
3.tgz
-rw-r--r--. 1 root root  157937478 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_f7ba33a6a9db9cfc4b0ab0f368569e19b9fa08f4c01a0d5f6a243d61ab781bd
8.tgz
-rw-r--r--. 1 root root  145535253 Oct 31 15:26 ocp-v4.0-art-
dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de
5.tgz
-rw-r--r--. 1 root root  158048761 Oct 31 15:40 ocp-v4.0-art-
dev@sha256_f914228ddbb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb4
7.tgz
-rw-r--r--. 1 root root  167914526 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_fa3ca9401c7a9efda0502240aeb8d3ae2d239d38890454f17fe5158b6230501
0.tgz
-rw-r--r--. 1 root root  164432422 Oct 31 15:24 ocp-v4.0-art-
dev@sha256_fc4783b446c70df30b3120685254b40ce13ba6a2b0bf8fb1645f116cf6a392f1
.tgz
-rw-r--r--. 1 root root  306643814 Oct 31 15:11
troubleshoot@sha256_b86b8aea29a818a9c22944fd18243fa0347c7a2bf1ad8864113ff2b
b2d8e0726.tgz
```

14.4.4. Operator イメージのダウンロード

また、5G 無線アクセスネットワーク (RAN) 分散ユニット (DU) クラスター設定で使用される Day-2 Operator を事前キャッシュすることもできます。Day-2 Operator は、インストールされている OpenShift Container Platform のバージョンに依存します。



重要

factory-precaching-cli ツールが RHACM およびマルチクラスターエンジン Operator の適切なコンテナーイメージを事前キャッシュできるように、--acm-version および --mce-version フラグを使用して、RHACM ハブおよびマルチクラスターエンジン Operator のバージョンを含める必要があります。

手順

- Operator イメージを事前キャッシュします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
```

```
--acm-version 2.6.3 \ 3
--mce-version 2.1.4 \ 4
-f /mnt \ 5
--img quay.io/custom/repository 6
--du-profile -s 7
```

1 factory-precaching-cli ツールのダウンロード機能を指定します。

2 OpenShift Container Platform リリースバージョンを定義します。

3 RHACM バージョンを定義します。

4 マルチクラスターエンジンのバージョンを定義します。

5 ディスク上のイメージをダウンロードするフォルダーを定義します。

6 オプション: 追加のイメージを保存するリポジトリーを定義します。これらのイメージはダウンロードされ、ディスクに事前キャッシュされます。

7 DU 設定に含まれる Operator の事前キャッシュを指定します。

出力例

```
Generated /mnt/imageset.yaml
Generating list of pre-cached artifacts...
Processing artifact [1/379]: ocp-v4.0-art-
dev@sha256_7753a8d9dd5974be8c90649aadd7c914a3d8a1f1e016774c7ac7c9422e9f99
58
Processing artifact [2/379]: ose-kube-rbac-
proxy@sha256_c27a7c01e5968aff16b6bb6670423f992d1a1de1a16e7e260d12908d33224
31c
```

```
Processing artifact [3/379]: ocp-v4.0-art-
dev@sha256_370e47a14c798ca3f8707a38b28cfc28114f492bb35fe1112e55d1eb51022c9
9
...
Processing artifact [378/379]: ose-local-storage-
operator@sha256_0c81c2b79f79307305e51ce9d3837657cf9ba5866194e464b4d1b299f85
034d0
Processing artifact [379/379]: multicluster-operators-channel-
rhel8@sha256_c10f6bbb84fe36e05816e873a72188018856ad6aac6cc16271a1b3966f73ce
b3
...
Summary:

Release:                 4.16.0
Hub Version:               2.6.3
ACM Version:               2.6.3
MCE Version:               2.1.4
Include DU Profile:          Yes
Workers:                  83
```

14.4.5. 非接続環境でのカスタムイメージの事前キャッシュ

--generate-imageset 引数は、ImageSetConfiguration カスタムリソース (CR) が生成された後に factory-precaching-cli ツールを停止します。これにより、イメージをダウンロードする前に ImageSetConfiguration CR をカスタマイズできます。CR をカスタマイズしたら、--skip-imageset 引数を使用して、ImageSetConfiguration CR で指定したイメージをダウンロードできます。

次の方法で ImageSetConfiguration CR をカスタマイズできます。

- Operator と追加のイメージを追加
- Operator と追加のイメージを削除
- Operator とカタログソースをローカルまたは切断されたレジストリーに変更

手順

1. イメージを事前キャッシュします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s \ 7
  --generate-imageset 8
```

1 factory-precaching-cli ツールのダウンロード機能を指定します。

2 OpenShift Container Platform リリースバージョンを定義します。

3 RHACM バージョンを定義します。

4 マルチクラスターエンジンのバージョンを定義します。

5 ディスク上のイメージをダウンロードするフォルダーを定義します。

6 オプション: 追加のイメージを保存するリポジトリーを定義します。これらのイメージはダウンロードされ、ディスクに事前キャッシュされます。

7 DU 設定に含まれる Operator の事前キャッシュを指定します。

8 --generate-imageset 引数は ImageSetConfiguration CR のみを生成します。これにより、CR をカスタマイズできます。

出力例

```
Generated /mnt/imageset.yaml
```

ImageSetConfiguration CR の例

```
apiVersion: mirror.openshift.io/v1alpha2
kind: ImageSetConfiguration
mirror:
  platform:
    channels:
    - name: stable-4.16
      minVersion: 4.16.0 (1)
      maxVersion: 4.16.0
  additionalImages:
    - name: quay.io/custom/repository
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.16
      packages:
        - name: advanced-cluster-management (2)
          channels:
            - name: 'release-2.6'
              minVersion: 2.6.3
              maxVersion: 2.6.3
        - name: multicluster-engine (3)
          channels:
            - name: 'stable-2.1'
              minVersion: 2.1.4
              maxVersion: 2.1.4
        - name: local-storage-operator (4)
          channels:
            - name: 'stable'
        - name: ptp-operator (5)
          channels:
            - name: 'stable'
        - name: sriov-network-operator (6)
          channels:
            - name: 'stable'
        - name: cluster-logging (7)
          channels:
            - name: 'stable'
        - name: lvms-operator (8)
          channels:
            - name: 'stable-4.16'
        - name: amq7-interconnect-operator (9)
          channels:
            - name: '1.10.x'
```

```
    - name: bare-metal-event-relay 10
      channels:
       - name: 'stable'
  - catalog: registry.redhat.io/redhat/certified-operator-index:v4.16
    packages:
     - name: sriov-fec 11
       channels:
        - name: 'stable'
```

1

プラットフォームのバージョンは、ツールに渡されたバージョンと一致します。

2 3

RHACM とマルチクラスターエンジン Operator のバージョンは、ツールに渡されるバージョンと一致します。

4 5 6 7 8 9 10 11

CR には、指定されたすべての DU Operator が含まれます。

2. CR でカタログリソースをカスタマイズします。

```
apiVersion: mirror.openshift.io/v1alpha2
kind: ImageSetConfiguration
mirror:
  platform:
[...]
  operators:
    - catalog: eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/certified-operator-
index:v4.16
      packages:
       - name: sriov-fec
         channels:
          - name: 'stable'
```

ローカルレジストリーまたは接続されていないレジストリーを使用してイメージをダウンロードする場合は、最初に、コンテンツの取得元のレジストリーの証明書を追加する必要があります。

3. エラーを回避するには、レジストリー証明書をサーバーにコピーします。

```
# cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
```

4. 次に、証明書トラストストアを更新します。

```
# update-ca-trust
```

5. ホストの /etc/pki フォルダーを factory-cli イメージにマウントします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged
--rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli download \ 1
  -r 4.16.0 \ 2
  --acm-version 2.6.3 \ 3
  --mce-version 2.1.4 \ 4
  -f /mnt \ 5
  --img quay.io/custom/repository 6
  --du-profile -s \ 7
  --skip-imageset 8
```

1 factory-precaching-cli ツールのダウンロード機能を指定します。

2 OpenShift Container Platform リリースバージョンを定義します。

3 RHACM バージョンを定義します。

4 マルチクラスターエンジンのバージョンを定義します。

5 ディスク上のイメージをダウンロードするフォルダーを定義します。

6 オプション: 追加のイメージを保存するリポジトリーを定義します。これらのイメージはダウンロードされ、ディスクに事前キャッシュされます。

7 DU 設定に含まれる Operator の事前キャッシュを指定します。

8 --skip-imageset 引数を使用すると、カスタマイズした ImageSetConfiguration CR で指定したイメージをダウンロードできます。

6. 新しい imageSetConfiguration CR を生成せずにイメージをダウンロードします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker --privileged --rm
quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli download -r 4.16.0
\
--acm-version 2.6.3 --mce-version 2.1.4 -f /mnt \
--img quay.io/custom/repository \
--du-profile -s \
--skip-imageset
```

関連情報

- オンラインの Red Hat レジストリーにアクセスするには、OpenShift インストールカスタマイズツール を参照してください。
- マルチクラスターエンジンの使用について、詳しくは マルチクラスターエンジン Operator を使用したクラスターのライフサイクル を参照してください。

14.5. GITOPS ZTP でのイメージの事前キャッシュ

SiteConfig マニフェストは、OpenShift クラスターをインストールおよび設定する方法を定義します。GitOps Zero Touch Provisioning (ZTP) プロビジョニングワークフローの場合、factory-precaching-cli ツールでは SiteConfig マニフェストに次の追加フィールドが必要です。

- clusters.ignitionConfigOverride
- nodes.installerArgs
- nodes.ignitionConfigOverride

追加フィールドを含む SiteConfig の例

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-5g-lab"
  namespace: "example-5g-lab"
spec:
  baseDomain: "example.domain.redhat.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "img4.9.10-x86-64-appsub" 1
  sshPublicKey: "ssh-rsa ..."
  clusters:
  - clusterName: "sno-worker-0"
    clusterImageSetNameRef: "eko4-img4.11.5-x86-64-appsub" 2
    clusterLabels:
      group-du-sno: ""
      common-411: true
      sites : "example-5g-lab"
      vendor: "OpenShift"
    clusterNetwork:
      - cidr: 10.128.0.0/14
        hostPrefix: 23
    machineNetwork:
      - cidr: 10.19.32.192/26
    serviceNetwork:
      - 172.30.0.0/16
    networkType: "OVNKubernetes"
    additionalNTPSources:
      - clock.corp.redhat.com
    ignitionConfigOverride:
      '{
        "ignition": {
          "version": "3.1.0"
        },
        "systemd": {
          "units": [
            {
              "name": "var-mnt.mount",
              "enabled": true,
              "contents": "[Unit]\nDescription=Mount partition with artifacts\nBefore=precache-
images.service\nBindsTo=precache-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precach
e-images.service"
            },
            {
              "name": "precache-images.service",
              "enabled": true,
              "contents": "[Unit]\nDescription=Extracts the precached images in discovery
stage\nAfter=var-
mnt.mount\nBefore=agent.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory
```

```
=/var/mnt\nExecStart=bash /usr/local/bin/extract-
ai.sh\n#TimeoutStopSec=30\n\n[Install]\nWantedBy=multi-user.target
default.target\nWantedBy=agent.service"
          }
        ]
      },
      "storage": {
        "files": [
          {
            "overwrite": true,
            "path": "/usr/local/bin/extract-ai.sh",
            "mode": 755,
            "user": {
              "name": "root"
            },
            "contents": {
              "source":
"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3
A-ai-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER
%0A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%
20total%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%2
0%27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%2
0%7C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%
20%24uri%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20exi
sting%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7
Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%
20%20%20%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20
%20%20%20%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri
%22%20%7C%20%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%
20%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20
%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20
copy%20dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29
%29%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEA
SE_LIST%7D%0A%0A%23%20workaround%20while%20https%3A%2F%2Fgithub.com%2Fope
nshift%2Fassisted-service%2Fpull%2F3546%0A%23cp%20%2Fvar%2Fmnt%2Fmodified-
rhcos-4.10.3-x86_64-metal.x86_64.raw.gz%20%2Fvar%2Ftmp%2F.%0A%0Aexit%200"
            }
          },
          {
            "overwrite": true,
            "path": "/usr/local/bin/agent-fix-bz1964591",
            "mode": 755,
            "user": {
              "name": "root"
```

```
            },
            "contents": {
              "source":
"data:,%23%21%2Fusr%2Fbin%2Fsh%0A%0A%23%20This%20script%20is%20a%20workarou
nd%20for%20bugzilla%201964591%20where%20symlinks%20inside%20%2Fvar%2Flib%2Fcon
tainers%2F%20get%0A%23%20corrupted%20under%20some%20circumstances.%0A%23%0A
%23%20In%20order%20to%20let%20agent.service%20start%20correctly%20we%20are%20che
cking%20here%20whether%20the%20requested%0A%23%20container%20image%20exists%2
0and%20in%20case%20%22podman%20images%22%20returns%20an%20error%20we%20try
%20removing%20the%20faulty%0A%23%20image.%0A%23%0A%23%20In%20such%20a%20s
cenario%20agent.service%20will%20detect%20the%20image%20is%20not%20present%20and
%20pull%20it%20again.%20In%20case%0A%23%20the%20image%20is%20present%20and%2
0can%20be%20detected%20correctly%2C%20no%20any%20action%20is%20required.%0A%0
AIMAGE%3D%24%28echo%20%241%20%7C%20sed%20%27s%2F%3A.%2A%2F%2F%27%29
%0Apodman%20image%20exists%20%24IMAGE%20%7C%7C%20echo%20%22already%20loa
ded%22%20%7C%7C%20echo%20%22need%20to%20be%20pulled%22%0A%23podman%20i
mages%20%7C%20grep%20%24IMAGE%20%7C%7C%20podman%20rmi%20--
force%20%241%20%7C%7C%20true"
            }
          }
        ]
      }
    }'
  nodes:
    - hostName: "snonode.sno-worker-0.example.domain.redhat.com"
      role: "master"
      bmcAddress: "idrac-
virtualmedia+https://10.19.28.53/redfish/v1/Systems/System.Embedded.1"
      bmcCredentialsName:
        name: "worker0-bmh-secret"
      bootMACAddress: "e4:43:4b:bd:90:46"
      bootMode: "UEFI"
      rootDeviceHints:
        deviceName: /dev/disk/by-path/pci-0000:01:00.0-scsi-0:2:0:0
      installerArgs: '["--save-partlabel", "data"]'
      ignitionConfigOverride: |
        {
          "ignition": {
            "version": "3.1.0"
          },
          "systemd": {
            "units": [
              {
                "name": "var-mnt.mount",
                "enabled": true,
                "contents": "[Unit]\nDescription=Mount partition with artifacts\nBefore=precache-
ocp-images.service\nBindsTo=precache-ocp-
images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-
partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=precach
e-ocp-images.service"
              },
              {
                "name": "precache-ocp-images.service",
                "enabled": true,
                "contents": "[Unit]\nDescription=Extracts the precached OCP images into
containers storage\nAfter=var-mnt.mount\nBefore=machine-config-daemon-pull.service
```

```
nodeip-
configuration.service\n\n[Service]\nType=oneshot\nUser=root\nWorkingDirectory=/var/mnt\nE
xecStart=bash /usr/local/bin/extract-
ocp.sh\nTimeoutStopSec=60\n\n[Install]\nWantedBy=multi-user.target"
              }
             ]
            },
            "storage": {
             "files": [
              {
               "overwrite": true,
               "path": "/usr/local/bin/extract-ocp.sh",
               "mode": 755,
               "user": {
                "name": "root"
               },
               "contents": {
                "source":
"data:,%23%21%2Fbin%2Fbash%0A%0AFOLDER%3D%22%24%7BFOLDER%3A-
%24%28pwd%29%7D%22%0AOCP_RELEASE_LIST%3D%22%24%7BOCP_RELEASE_LIST%3
A-ocp-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER
%0A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%24OCP_RELEASE_LIST%20%7C%20wc%20-
l%29%20%20%23%20Required%20to%20keep%20track%20of%20the%20pull%20task%20vs%
20total%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%2
0%27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%2
0%7C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%
20%24uri%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20%20echo%20%22Skipping%20exi
sting%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7
Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%
20%20%20%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20
%20%20%20%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri
%22%20%7C%20%20rev%20%7C%20cut%20-d%20%22%2F%22%20-
f1%20%7C%20rev%20%7C%20tr%20%22%3A%22%20%22_%22%29%0A%20%20tar%20zxvf%
20%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20
%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20
copy%20dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29
%29%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEA
SE_LIST%7D%0A%0Aexit%200"
              }
             }
            ]
           }
          }
        nodeNetwork:
         config:
          interfaces:
           - name: ens1f0
```

```
        type: ethernet
        state: up
        macAddress: "AA:BB:CC:11:22:33"
        ipv4:
          enabled: true
          dhcp: true
        ipv6:
          enabled: false
      interfaces:
        - name: "ens1f0"
          macAddress: "AA:BB:CC:11:22:33"
```

1

spec.clusters.clusterImageSetNameRef フィールドに別のイメージセットを指定しない限り、デプロイメントに使用されるクラスターイメージセットを指定します。

2

個々のクラスターをデプロイするために使用されるクラスターイメージセットを指定します。定義されている場合には、サイトレベルで spec.clusterImageSetNameRef を上書きします。

14.5.1. clusters.ignitionConfigOverride フィールドについて

clusters.ignitionConfigOverride フィールドは、GitOps ZTP 検出段階で Ignition 形式の設定を追加します。この設定には、仮想メディアにマウントされた ISO の systemd サービスが含まれます。これにより、スクリプトが検出 RHCOS ライブ ISO の一部となり、アシステッドインストーラー (AI) イメージのロードにスクリプトを使用できるようになります。

systemd サービス

systemd サービスは var-mnt.mount と precache-images.services です。precache-images.service は、var-mnt.mount ユニットによって /var/mnt にマウントされるディスクパーティションに依存します。このサービスは、extract-ai.sh というスクリプトを呼び出します。

extract-ai.sh

extract-ai.sh スクリプトは、必要なイメージをディスクパーティションからローカルコンテナーストレージに展開してロードします。スクリプトが正常に終了したら、イメージをローカルで使用できます。

agent-fix-bz1964591

agent-fix-bz1964591 スクリプトは、AI の問題の回避策です。AI がイメージを削除して、agent.service がレジストリーからイメージを再度プルするように強制するのを防ぐため

に、agent-fix-bz1964591 スクリプトは、要求されたコンテナーイメージが存在するかどうかを確認します。

14.5.2. nodes.installerArgs フィールドについて

nodes.installerArgs フィールドでは、coreos-installer ユーティリティーが RHCOS ライブ ISO をディスクに書き込む方法を設定できます。data とラベル付けされたディスクパーティションを保存するよう指定する必要があります。これは、data パーティションに保存されたアーティファクトが OpenShift Container Platform のインストール段階で必要になるためです。

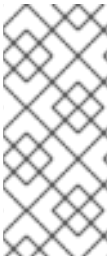
追加のパラメーターは、ライブ RHCOS をディスクに書き込む coreos-installer ユーティリティーに直接渡されます。次回の再起動時に、オペレーティングシステムはディスクから起動します。

coreos-installer ユーティリティーには、いくつかのオプションを渡すことができます。

```
OPTIONS:
...
    -u, --image-url <URL>
            Manually specify the image URL

    -f, --image-file <path>
            Manually specify a local image file

    -i, --ignition-file <path>
            Embed an Ignition config from a file

    -I, --ignition-url <URL>
            Embed an Ignition config from a URL
...
        --save-partlabel <lx>...
            Save partitions with this label glob

        --save-partindex <id>...
            Save partitions with this number or range
...
        --insecure-ignition
            Allow Ignition URL without HTTPS or hash
```

14.5.3. nodes.ignitionConfigOverride フィールドについて

clusters.ignitionConfigOverride と同様に、nodes.ignitionConfigOverride フィールドを使用すると、Ignition 形式の設定を coreos-installer ユーティリティーに追加できます。ただし、これを追加できるのは、OpenShift Container Platform のインストール段階です。RHCOS がディスクに書き込まれると、GitOps ZTP 検出 ISO に含まれる追加の設定は使用できなくなります。検出段階で、追加の設定はライブ OS のメモリーに保存されます。



注記

この段階では、展開およびロードされたコンテナーイメージの数は、検出段階よりも多くなります。OpenShift Container Platform のリリースと、Day-2 Operators をインストールするかどうかによって、インストール時間は異なります。

インストール段階では、var-mnt.mount および precache-ocp.services systemd サービスが使用されます。

precache-ocp.service

precache-ocp.service は、var-mnt.mount ユニットによって /var/mnt にマウントされるディスクパーティションに依存します。precache-ocp.service サービスは、extract-ocp.sh というスクリプトを呼び出します。

重要

OpenShift Container Platform のインストール前にすべてのイメージを展開するには、machine-config-daemon-pull.service および nodeip-configuration.service サービスを実行する前に precache-ocp.service を実行する必要があります。

extract-ocp.sh

extract-ocp.sh スクリプトは、必要なイメージをディスクパーティションからローカルコンテナーストレージに展開してロードします。

SiteConfig とオプションの PolicyGenerator または PolicyGenTemplate カスタムリソース(CR)を、Argo CD が監視している Git リポジトリーにコミットすると、CR をハブクラスターと同期することで GitOps ZTP ワークフローを開始できます。

14.6. RENDERED CATALOG IS INVALID のトラブルシューティング

ローカルまたは非接続レジストリーを使用してイメージをダウンロードすると、The rendered catalog is invalid というエラーが表示される場合があります。これは、コンテンツの取得元である新しいレジストリーの証明書が不足していることを意味します。

注記

factory-precaching-cli ツールイメージは、UBI RHEL イメージ上に構築されています。証明書のパスと場所は RHCOS でも同じです。

エラーの例

```
Generating list of pre-cached artifacts...
error: unable to run command oc-mirror -c /mnt/imageset.yaml file:///tmp/fp-cli-
3218002584/mirror --ignore-history --dry-run: Creating directory: /tmp/fp-cli-
3218002584/mirror/oc-mirror-workspace/src/publish
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/v2
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/charts
Creating directory: /tmp/fp-cli-3218002584/mirror/oc-mirror-workspace/src/release-signatures
backend is not configured in /mnt/imageset.yaml, using stateless mode
backend is not configured in /mnt/imageset.yaml, using stateless mode
No metadata detected, creating new workspace
level=info msg=trying next host error=failed to do request: Head
"https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-
index/manifests/v4.11": x509: certificate signed by unknown authority
host=eko4.cloud.lab.eng.bos.redhat.com:8443

The rendered catalog is invalid.

Run "oc-mirror list operators --catalog CATALOG-NAME --package PACKAGE-NAME" for
more information.

error: error rendering new refs: render reference
"eko4.cloud.lab.eng.bos.redhat.com:8443/redhat/redhat-operator-index:v4.11": error resolving
name : failed to do request: Head
"https://eko4.cloud.lab.eng.bos.redhat.com:8443/v2/redhat/redhat-operator-
index/manifests/v4.11": x509: certificate signed by unknown authority
```

手順

1. レジストリー証明書をサーバーにコピーします。

```
# cp /tmp/eko4-ca.crt /etc/pki/ca-trust/source/anchors/.
```

2. 証明書トラストストアを更新します。

```
# update-ca-trust
```

3. ホストの /etc/pki フォルダーを factory-cli イメージにマウントします。

```
# podman run -v /mnt:/mnt -v /root/.docker:/root/.docker -v /etc/pki:/etc/pki --privileged
-it --rm quay.io/openshift-kni/telco-ran-tools:latest -- \
factory-precaching-cli download -r 4.16.0 --acm-version 2.5.4 \
  --mce-version 2.0.4 -f /mnt \--img quay.io/custom/repository
  --du-profile -s --skip-imageset
```

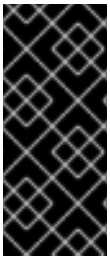
第15章 シングルノード OPENSHIFT クラスターのイメージベースのアップグレード

15.1. シングルノード OPENSHIFT クラスターのイメージベースのアップグレードについて

OpenShift Container Platform 4.14.13 以降、ライフサイクルエージェントは、単一ノードの OpenShift クラスターのプラットフォームバージョンをアップグレードする代替方法を提供します。イメージベースのアップグレードは標準のアップグレード方法よりも高速であり、OpenShift Container Platform <4.y> から <4.y+2>、および <4.y.z> から <4.y.z+n> に直接アップグレードできます。

このアップグレード方法では、ターゲットのシングルノード OpenShift クラスターにインストールされている専用のシードクラスターから、新しい ostree stateroot として生成された OCI イメージを利用します。シードクラスターは、ターゲットの OpenShift Container Platform バージョン、Day 2 Operator、およびすべてのターゲットクラスターに共通する単一ノードの OpenShift クラスターです。

シードクラスターから生成されたシードイメージを使用して、シードクラスターと同じハードウェア、Day 2 Operator、およびクラスター設定の組み合わせを持つシングルノード OpenShift クラスターで、プラットフォームバージョンをアップグレードできます。



重要

イメージベースのアップグレードでは、クラスターが実行されているハードウェアプラットフォームに固有のカスタムイメージが使用されます。それぞれのハードウェアプラットフォームには、個別のシードイメージが必要です。

Lifecycle エージェントは、参加するクラスターで 2 つのカスタムリソース(CR)を使用して、アップグレードをオーケストレーションします。

- シードクラスターでは、Browse dGenerator CR では、シード イメージ生成が可能です。この CR は、シードイメージをプッシュするリポジトリーを指定します。
- ターゲットクラスターでは、ImageBasedUpgrade CR は、ターゲットクラスターのアップグレード用のシードイメージとワークロードのバックアップ設定を指定します。

SeedGenerator CR の例

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

```
kind: SeedGenerator
metadata:
  name: seedimage
spec:
  seedImage: <seed_image>
```

ImageBasedUpgrade CR の例

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle 1
  seedImageRef: 2
    version: <target_version>
    image: <seed_container_image>
    pullSecretRef:
      name: <seed_pull_secret>
  autoRollbackOnFailure: {}
#    initMonitorTimeoutSeconds: 1800 3
  extraManifests: 4
  - name: example-extra-manifests
    namespace: openshift-lifecycle-agent
  oadpContent: 5
  - name: oadp-cm-example
    namespace: openshift-adp
```

1

ImageBasedUpgrade CR の必要なステージを定義します。値は、Idle、Prep、Upgrade、または Rollback にすることができます。

2

ターゲットプラットフォームのバージョン、使用するシードイメージ、およびイメージへのアクセスに必要なシークレットを定義します。

3

（オプション）最初の再起動後に、アップグレードが完了しない場合に、ロールバックする時間枠を秒単位で指定します。定義されていないか、0 に設定されている場合は、デフォルト値の

1800 秒(30 分)が使用されます。

4

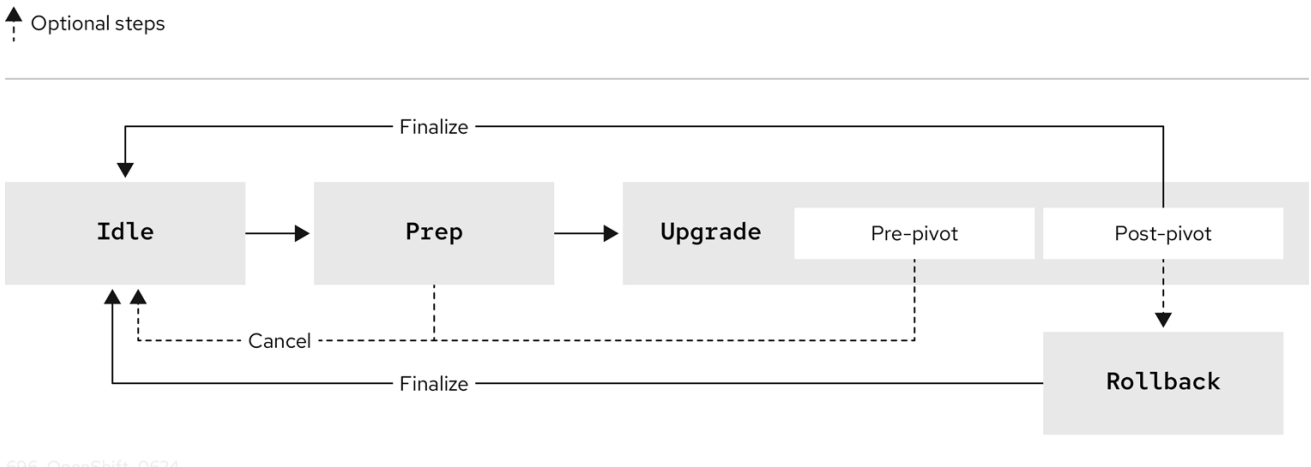
5

OADP Backup CR および Restore CR を含む ConfigMap リソースのリストを指定します。

15.1.1. イメージベースのアップグレードの段階

シードクラスターでシードイメージを生成した後、spec.stage フィールドを ImageBasedUpgrade CR の以下のいずれかの値に設定して、ターゲットクラスターのステージ間を移動できます。

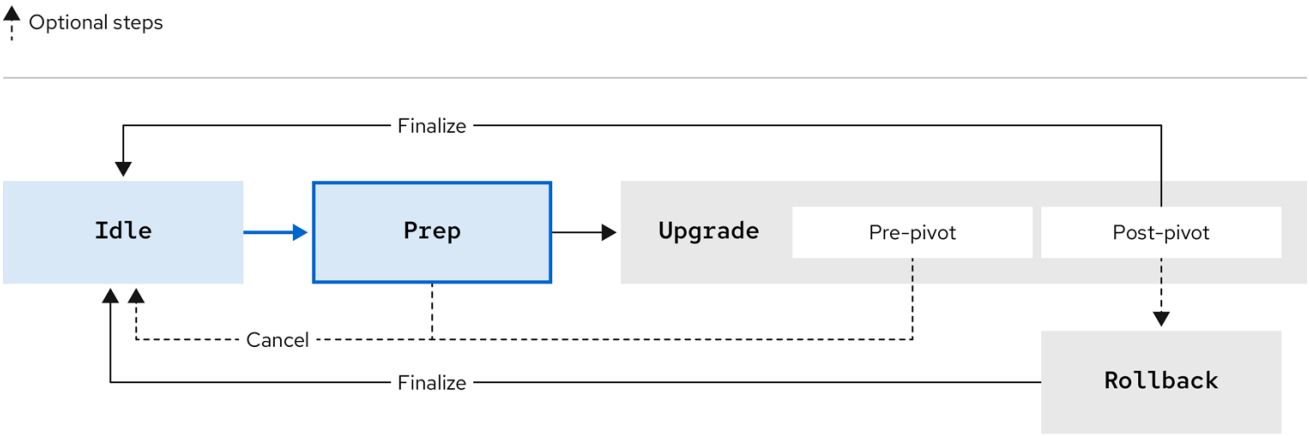
- --idle
- PReP
- アップグレード
- rollback （任意）



15.1.1.1. アイドルステージ

Lifecycle エージェントは、Operator の初回デプロイ時に stage: Idle に設定された ImageBasedUpgrade CR を作成します。これはデフォルトのスタックです。進行中のアップグレード

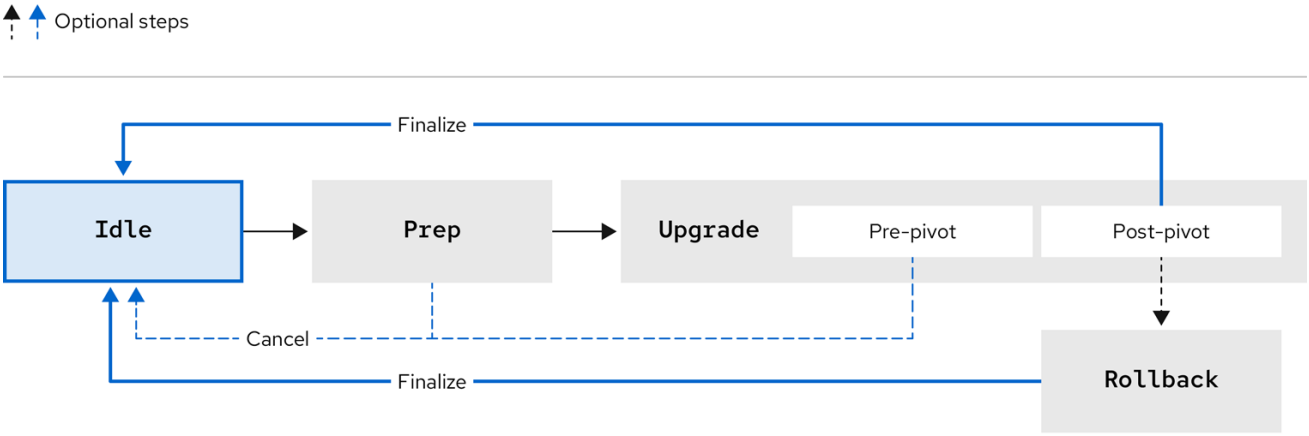
はなく、クラスターは Prep 段階に移行する準備ができています。

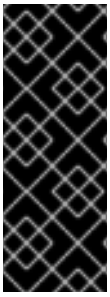


また、Idle 段階に移動して、次のいずれかの手順を実行します。

- アップグレードの正常な完了
- ロールバックの最終処理
- アップグレード ステージの事前コピーフェーズまで、進行中の アップグレード をキャンセルします。

Idle 段階に移行すると、ライフサイクルエージェントがリソースをクリーンアップし、クラスターが再びアップグレードの準備を整えるようにします。

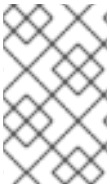




重要

アップグレードをキャンセルするときに RHACM を使用する場合は、ターゲットのマネージドクラスターから import.open-cluster-management.io/disable-auto-import アノテーションを削除して、クラスターの自動インポートを再度有効にする必要があります。

15.1.1.2. PReP ステージ



注記

スケジュールされたメンテナンス期間の前にこの段階を完了できます。

Prep ステージでは、ImageBasedUpgrade CR で次のアップグレードの詳細を指定します。

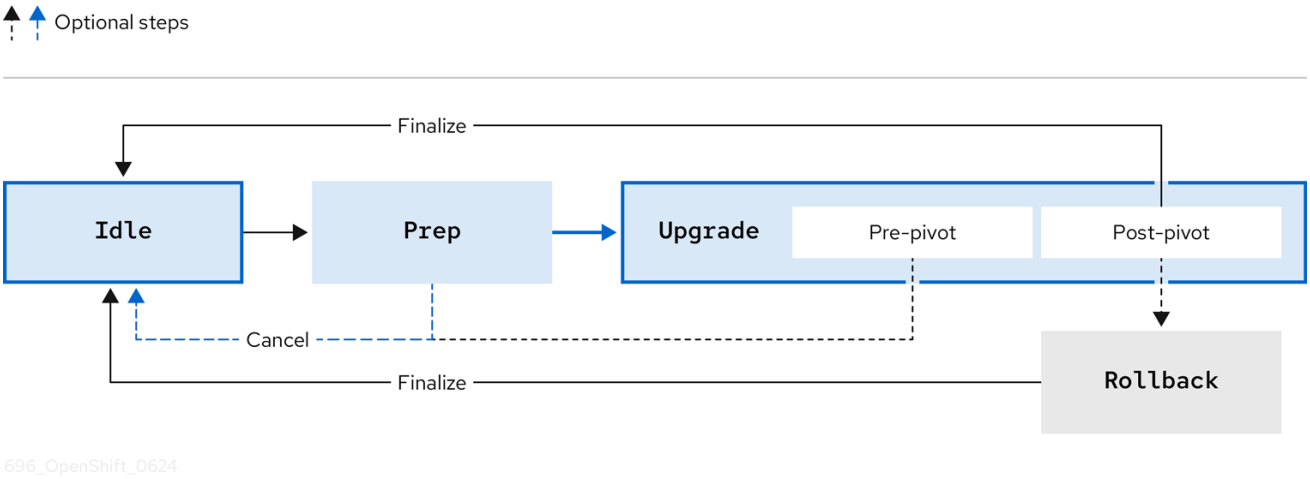
- 使用するシードイメージ
- バックアップするリソース
- アップグレード後も保持する追加のマニフェストおよびカスタムカタログソース（存在する場合）

次に、指定した内容に基づいて、ライフサイクルエージェントは、現在実行中のバージョンに影響を与えることなく、アップグレードを準備します。この段階で、ライフサイクルエージェントは、特定の条件が満たされているかどうかを確認して、ターゲットクラスターが Upgrade 段階に進むようにし、シードイメージで指定された追加のコンテナーイメージと共に、シードイメージをターゲットクラスターにプルします。

また、OADP Operator の Backup および Restore CR を使用してバックアップリソースを準備します。これらの CR は Upgrade 段階で使用され、クラスターを再設定し、RHACM に登録し、アプリケーションアーティファクトを復元します。

OADP Operator に加えて、ライフサイクルエージェントは ostree バージョンシステムを使用してバックアップを作成します。これにより、アップグレードとロールバックの両方後にクラスターの再設定を完了できます。

Prep 段階が完了したら、Idle 段階に移行することでアップグレードプロセスをキャンセルするか、ImageBasedUpgrade CR の Upgrade ステージに移動してアップグレードを開始することができます。アップグレードをキャンセルすると、Operator はクリーンアップ操作を実行します。



15.1.1.3. アップグレード段階

Upgrade ステージは、以下の 2 つのフェーズで設定されます。

pre-pivot

新しい状態ルートにピボットする前に、Lifecycle Agent は必要なクラスター固有のアーティファクトを収集し、新しい stateroot に保存します。Prep stage で指定されたクラスターリソースのバックアップは、互換性のあるオブジェクトストレージソリューションに作成されます。Lifecycle Agent は、ImageBasedUpgrade CR の extraManifests フィールドで指定された CR、またはターゲットクラスターにバインドされた ZTP ポリシーで説明されている CR をエクスポートします。コピー前のフェーズが完了すると、Lifecycle Agent は、新しい stateroot デプロイメントをデフォルトのブートエントリーとして設定し、ノードを再起動します。

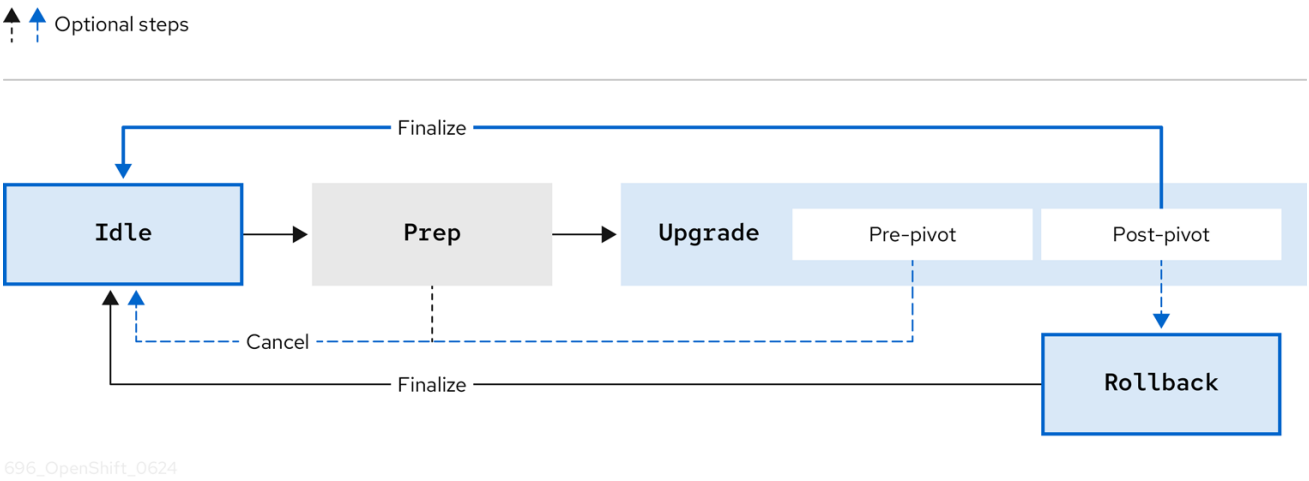
post-pivot

新しい状態ルートから起動した後、ライフサイクルエージェントは、コピー前のフェーズで収集されたクラスター固有のアーティファクトを適用してクラスターを再設定します。Operator は保存されたすべての CR を適用し、バックアップを復元します。Operator は、シードイメージのクラスター暗号も再生成します。これにより、同じシードイメージを使用してアップグレードされたシングルノード OpenShift クラスターには、一意で有効な暗号化オブジェクトが与えられます。

アップグレードが完了し、変更に満足したら、Idle ステージに移動してアップグレードを完了できます。

重要

アップグレードの最終処理後、元のリリースにロールバックすることはできません。



アップグレードをキャンセルする場合は、Upgrade ステージの事前コピーフェーズまでこれを実行できます。アップグレード後に問題が発生した場合は、手動のロールバックの Rollback ステージに移動できます。

15.1.1.4. （オプション）ロールバックステージ

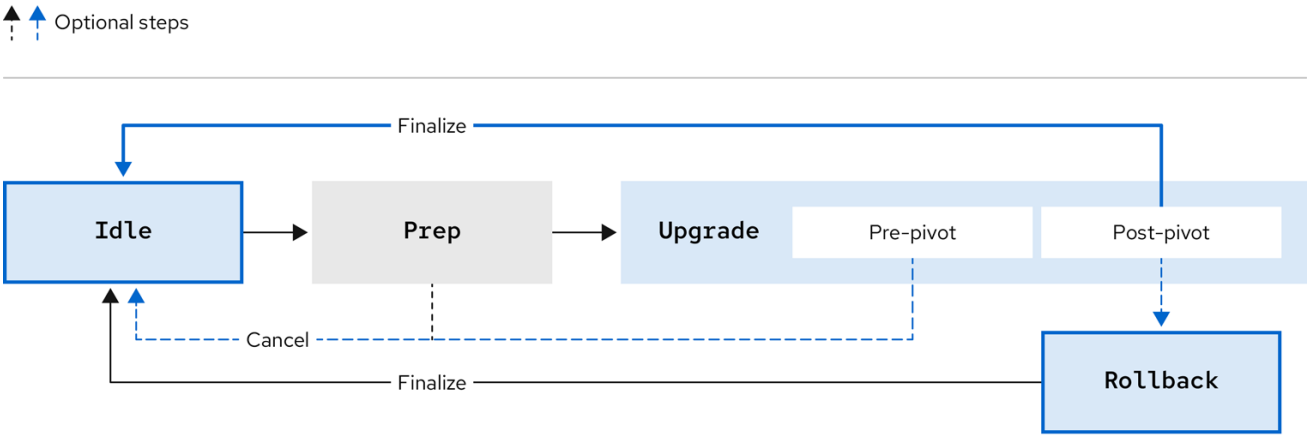
ロールバック ステージは、障害時に手動で、または自動で開始できます。Rollback ステージ中に、ライフサイクルエージェントは元の ostree stateroot デプロイメントをデフォルトとして設定します。次に、ノードは以前のリリースの OpenShift Container Platform およびアプリケーション設定で再起動します。

警告

ロールバック後に Idle 段階に移行すると、ライフサイクルエージェントは、失敗したアップグレードのトラブルシューティングに使用できるリソースをクリーンアップします。

指定した制限時間内にアップグレードが完了しないと、ライフサイクルエージェントが自動ロールバックを開始します。自動ロールバックの詳細は、関連 （オプション）Lifecycle Agent を使用した

ロールバックの開始 セクションを参照してください。



関連情報

- Lifecycle Agent を使用したイメージベースのアップグレードの実行
- Lifecycle Agent と GitOps ZTP を使用したイメージベースのアップグレードの実行
- （オプション）Lifecycle Agent を使用したイメージベースのアップグレードのロールバックステージへの移行
- （オプション）ライフサイクルエージェントと GitOps ZTP を使用したイメージベースのアップグレードのロールバックステージへの移行

15.1.2. イメージベースのアップグレードのガイドライン

イメージベースのアップグレードを成功させるには、デプロイメントは特定の要件を満たす必要があります。

イメージベースのアップグレードを実行するには、さまざまなデプロイメント方法があります。

GitOps ZTP

GitOps Zero Touch Provisioning (ZTP)を使用して、クラスターをデプロイおよび設定します。

Non-GitOps

Red Hat Advanced Cluster Management (RHACM)を使用してクラスターをデプロイおよび設定するだけです。

切断された環境でイメージベースのアップグレードを実行できます。切断された環境のイメージをミラーリングする方法は、非接続インストールのイメージのミラーリングを参照してください。

関連情報

- 非接続インストールのイメージのミラーリング

15.1.2.1. コンポーネントの最小ソフトウェアバージョン

デプロイ方法によっては、イメージベースのアップグレードには、以下に示す最小限のソフトウェアバージョンが必要です。

表15.1 コンポーネントの最小ソフトウェアバージョン

コンポーネント	ソフトウェアバージョン	必須
Lifecycle Agent	4.16	はい
OADP Operator	1.3.1	はい
マネージドクラスターのバージョン	4.14.13	はい
ハブクラスターのバージョン	4.16	Yes (RHACM を使用している場合)
RHACM	2.10.2	Yes (RHACM を使用している場合)
GitOps ZTP プラグイン	4.16	GitOps ZTP デプロイメント方法のみ
Red Hat OpenShift GitOps	1.12	GitOps ZTP デプロイメント方法のみ
Topology Aware Lifecycle Manager (TALM)	4.16	GitOps ZTP デプロイメント方法のみ

コンポーネント	ソフトウェアバージョン	必須
Local Storage Operator	4.14	はい
Logical Volume Manager (LVM) Storage	4.14.2	はい

1. 永続ストレージは、両方ではなく、LVM Storage またはローカルストレージ Operator のいずれかによって提供される必要があります。

15.1.2.2. ハブクラスターのガイドライン

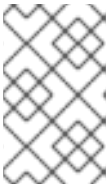
Red Hat Advanced Cluster Management (RHACM)を使用している場合は、ハブクラスターが次の条件を満たす必要があります。

- シードイメージに RHACM リソースを含めないようにするには、シードイメージを生成する前に、すべてのオプションの RHACM アドオンを無効にする必要があります。
- ハブクラスターは、ターゲット単一ノードの OpenShift クラスターをターゲットベースのアップグレードを実行する前に、少なくともターゲットバージョンにアップグレードする必要があります。

15.1.2.3. シードイメージのガイドライン

シードイメージは、同様の設定を持つ単一ノード OpenShift クラスターのセットをターゲットにします。つまり、シードクラスターには次の項目のターゲットクラスターと同じ設定が必要です。

- パフォーマンスプロファイル
- ターゲットクラスターの MachineConfig リソース
- --ip-version {4,6}



注記

本リリースでは、デュアルスタックネットワークはサポート対象外です。

- Lifecycle Agent および OADP Operator を含む Day 2 Operator の設定
- <disconnected_registry>
- FIPS 設定
- ターゲットクラスターに複数の IP があり、そのうちの 1 つがシードイメージの作成に使用されるサブネットに属する場合、ターゲットクラスターのノード IP がそのサブネットに属していない場合は、アップグレードに失敗します。

以下の設定は、参加するクラスターについて部分的にのみ一致する必要があります。

- ターゲットクラスターにプロキシー設定がある場合、シードクラスターにはプロキシー設定が必要ですが、設定は同じである必要はありません。
- コンテナーストレージ用のプライマリーディスクの専用パーティションは、参加するすべてのクラスターに必要です。ただし、パーティションのサイズと開始は同じである必要はありません。MachineConfig CR の spec.config.storage.disks.partitions.label: varlibcontainers ラベルのみが、シードクラスターとターゲットクラスターの両方で一致する必要があります。ディスクパーティションを作成する方法の詳細については、「ostree stateroots 間の共有コンテナーディレクトリーの設定」または「GitOps ZTP 使用時の ostree stateroots 間の共有コンテナーディレクトリーの設定」を参照してください。

シードイメージに含める内容の詳細については、Seed image configuration および RAN DU プロファイルを使用した Seed image configuration を参照してください。

関連情報

- ostree stateroots 間で共有コンテナーディレクトリーの設定
- GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設

定

- シードイメージ設定

15.1.2.4. OADP のバックアップと復元のガイドライン

OADP Operator を使用すると、ConfigMap オブジェクトでラップされた Backup CR および Restore CR を使用して、ターゲットクラスターでアプリケーションをバックアップおよび復元できます。アプリケーションは、アップグレード後に復元できるように、現行およびターゲットの OpenShift Container Platform バージョンで動作する必要があります。バックアップには、最初に作成されたリソースを含める必要があります。

次のリソースをバックアップから除外する必要があります。

- pods
- endpoints
- controllerrevision
- podmetrics
- packagemanifest
- replicaset
- LocalVolume (ローカル ストレージ Operator (LSO)を使用している場合)

単一ノード OpenShift には 2 つのローカルストレージ実装があります。

Local Storage Operator (LSO)

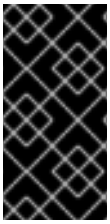
ライフサイクルエージェントは、LocalVolume およびそれに関連付けられた StorageClass を

含む必要なアーティファクトをバックアップおよび復元し ます。アプリケーションの Backup CR の persistentVolumes リソースを除外する必要があります。

LVM Storage

LVM ストレージアーティファクトの Backup CR および Restore CR を作成する必要があります。アプリケーションの Backup CR に persistentVolumes リソースを含める必要があります。

イメージベースのアップグレードでは、指定されたターゲットクラスターでサポートされる Operator は 1 つだけです。



重要

両方の Operator については、ImageBasedUpgrade CR を介して追加のマニフェストとして Operator CR を適用しないでください。

永続ボリュームの内容は、ピボット後に保持され、使用されます。DataProtectionApplication CR を設定する場合は、イメージベースのアップグレードのために .spec.configuration.restic.enable が false に設定されていることを確認する必要があります。これにより、Container Storage Interface の統合が無効になります。

15.1.2.4.1. lca.openshift.io/apply-wave ガイドライン

lca.openshift.io/apply-wave アノテーションは、Backup または Restore CR の適用順序を決定します。アノテーションの値は文字列番号である必要があります。Backup CR または Restore CR で lca.openshift.io/apply-wave アノテーションを定義する場合、それらはアノテーションの値に基づいて高い順序で適用されます。アノテーションを定義しない場合は、それらは一緒に適用されます。

lca.openshift.io/apply-wave アノテーションは、プラットフォームの Restore CR で数値的に低くする必要があります（例：RHACM および LVM ストレージアーティファクト）。このようにして、プラットフォームアーティファクトがアプリケーションの前に復元されます。

アプリケーションにクラススコープのリソースが含まれる場合、個別の Backup CR および Restore CR を作成し、バックアップをアプリケーションによって作成された特定のクラススコープのリソースに限定する必要があります。残りのアプリケーションの Restore CR の前に、クラススコープのリソースの Restore CR を復元する必要があります。

15.1.2.4.2. lca.openshift.io/apply-label ガイドライン

lca.openshift.io/apply-label アノテーションを使用して、特定のリソースのみをバックアップでき

ます。アノテーションで定義するリソースに基づいて、Lifecycle Agent は lca.openshift.io/backup: <backup_name > ラベルを適用し、 Backup CR の作成時に labelSelector.matchLabels.lca.openshift.io/backup: <backup_name > ラベルセレクターを指定されたリソースに追加します。

特定のリソースのバックアップに lca.openshift.io/apply-label アノテーションを使用するには、アノテーションに一覧表示されるリソースも spec セクションに追加する必要があります。lca.openshift.io/apply-label アノテーションが Backup CR で使用される場合は、他のリソースタイプが spec セクションで指定される場合でも、アノテーションに一覧表示されるリソースのみがバックアップされます。

CR の例:

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
  annotations:
    lca.openshift.io/apply-label:
rbac.authorization.k8s.io/v1/clusterroles/klusterlet,apps/v1/deployments/open-cluster-
management-agent/klusterlet (1)
  labels:
    velero.io/storage-location: default
spec:
  includedNamespaces:
   - open-cluster-management-agent
  includedClusterScopedResources:
   - clusterroles
  includedNamespaceScopedResources:
   - deployments
```

(1) 値は、クラスタースコープのリソースの group/version/resource/name 形式のコンマ区切りのオブジェクト、または namespace スコープリソースの group/version/resource/namespace/name 形式のコンマ区切りのオブジェクトのリストである必要があり、関連する Backup CR に割り当てる必要があります。

15.1.2.5. マニフェストの追加ガイドライン

Lifecycle Agent は、新しいデフォルトの stateroot デプロイメントで再起動した後、およびアプリケーションアーティファクトを復元する前に、追加のマニフェストを使用してターゲットクラスターを

復元します。

デプロイメント方法によって、追加のマニフェストを適用する別の方法が必要になります。

GitOps ZTP

lca.openshift.io/target-ocp-version: <target_ocp_version > ラベルを使用して、ライフサイクルエージェントがピボットの後に抽出して適用する必要がある追加マニフェストをマークします。ImageBasedUpgrade CR の lca.openshift.io/target-ocp-version -manifest-count アノテーションを使用して、lca.openshift.io/target-ocp-version-manifest-count でラベル付けされたマニフェストの数を指定できます。指定した場合、ライフサイクル環境は、ポリシーから抽出したマニフェストの数が、prep および upgrade ステージ中にアノテーションで指定された数と一致することを確認します。

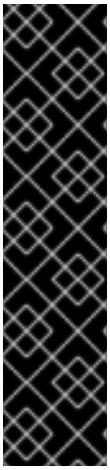
lca.openshift.io/target-ocp-version-manifest-count アノテーションの例

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  annotations:
    lca.openshift.io/target-ocp-version-manifest-count: "5"
  name: upgrade
```

Non-Gitops

追加マニフェストに lca.openshift.io/apply-wave アノテーションを使用してマークを付け、適用順序を判別します。ラベル付きの追加マニフェストは ConfigMap オブジェクトでラップされ、ピボット後に Lifecycle Agent が使用する ImageBasedUpgrade CR で参照されます。

ターゲットクラスターがカスタムカタログソースを使用する場合は、正しいリリースバージョンを参照する追加のマニフェストとしてそれらを含める必要があります。



重要

次の項目を追加マニフェストとして適用することはできません。

- MachineConfig オブジェクト
- OLM Operator のサブスクリプション

関連情報

- Lifecycle Agent を使用したイメージベースのアップグレードの実行
- Lifecycle Agent と GitOps ZTP を使用したイメージベースのアップグレードの実行
- ZTP 用のハブクラスターの準備
- Lifecycle Agent を使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成
- GitOps ZTP を使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成
- OADP のインストールについて

15.2. シングルノード OPENSHIFT クラスターのイメージベースのアップグレードの準備

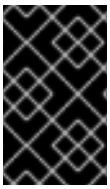
15.2.1. イメージベースのアップグレード用の共有コンテナーディレクトリーの設定

シングルノード OpenShift クラスターでは、イメージベースのアップグレード用に共有の var/lib/containers パーティションが必要です。これはインストール時に実行できます。

15.2.1.1. ostree stateroots 間で共有コンテナーディレクトリーの設定

インストール時に seed とターゲットクラスターの両方に MachineConfig を適用して、別のパー

ティションを作成し、アップグレードプロセスで使用される 2 つの ostree stateroot 間で /var/lib/containers ディレクトリーを共有します。



重要

この手順はインストール時に完了する必要があります。

手順

- MachineConfig を適用して、別のパーティションを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-containers-partitioned
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      disks:
        - device: /dev/disk/by-id/wwn-<root_disk> 1
          partitions:
            - label: varlibcontainers
              startMiB: <start_of_partition> 2
              sizeMiB: <partition_size> 3
      filesystems:
        - device: /dev/disk/by-partlabel/varlibcontainers
          format: xfs
          mountOptions:
            - defaults
            - prjquota
          path: /var/lib/containers
          wipeFilesystem: true
    systemd:
      units:
        - contents: |-
            # Generated by Butane
            [Unit]
            Before=local-fs.target
            Requires=systemd-fsck@dev-disk-by\x2dpartlabel-varlibcontainers.service
            After=systemd-fsck@dev-disk-by\x2dpartlabel-varlibcontainers.service

            [Mount]
            Where=/var/lib/containers
            What=/dev/disk/by-partlabel/varlibcontainers
            Type=xfs
            Options=defaults,prjquota
```

```
    [Install]
    RequiredBy=local-fs.target
  enabled: true
  name: var-lib-containers.mount
```

1

ルートディスクを指定します。

2

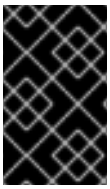
パーティションの開始点を MiB 単位で指定します。値が小さすぎると、インストールは失敗します。

3

パーティションの最小サイズを指定して、事前キャッシュされたイメージに十分なディスク領域を確保します。値が小さすぎると、インストール後のデプロイメントに失敗します。

15.2.1.2. GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設定

GitOps Zero Touch Provisioning (ZTP)ワークフローを使用している場合は、次の手順を実行して、シードクラスターとターゲットクラスターの両方に個別のディスクパーティションを作成し、/var/lib/containers ディレクトリーを共有します。



重要

この手順はインストール時に完了する必要があります。

前提条件

- Butane をインストールします。

手順

1. storage.bu ファイルを作成します。

```
variant: fcos
version: 1.3.0
storage:
  disks:
```

```
- device: /dev/disk/by-id/wwn-<root_disk> 1
  wipe_table: false
  partitions:
  - label: var-lib-containers
    start_mib: <start_of_partition> 2
    size_mib: <partition_size> 3
filesystems:
  - path: /var/lib/containers
    device: /dev/disk/by-partlabel/var-lib-containers
    format: xfs
    wipe_filesystem: true
    with_mount_unit: true
    mount_options:
      - defaults
      - prjquota
```

1 ルートディスクを指定します。

2 パーティションの開始点を MiB 単位で指定します。値が小さすぎると、インストールは失敗します。

3 パーティションの最小サイズを指定して、事前キャッシュされたイメージに十分なディスク領域を確保します。値が小さすぎると、インストール後のデプロイメントに失敗します。

2. storage.bu を Ignition ファイルに変換します。

```
$ butane storage.bu
```

出力例

```
{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62","partitions":[{"label":"var-lib-containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd":{"units":[{"contents":"# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
```

```
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}
```

3. 出力を SiteConfig CR の .spec.clusters.nodes.ignitionConfigOverride フィールドにコピーします。

```
[...]
spec:
  clusters:
    - nodes:
        - ignitionConfigOverride: '{"ignition":{"version":"3.2.0"},"storage":{"disks":
[{"device":"/dev/disk/by-id/wwn-0x6b07b250ebb9d0002a33509f24af1f62","partitions":
[{"label":"var-lib-
containers","sizeMiB":0,"startMiB":250000}],"wipeTable":false}],"filesystems":
[{"device":"/dev/disk/by-partlabel/var-lib-containers","format":"xfs","mountOptions":
["defaults","prjquota"],"path":"/var/lib/containers","wipeFilesystem":true}]},"systemd"
:{"units":[{"contents":"# Generated by Butane\n[Unit]\nRequires=systemd-fsck@dev-
disk-by\\x2dpartlabel-var\\x2dlib\\x2dcontainers.service\nAfter=systemd-fsck@dev-
disk-by\\x2dpartlabel-
var\\x2dlib\\x2dcontainers.service\n\n[Mount]\nWhere=/var/lib/containers\nWhat=/dev/
disk/by-partlabel/var-lib-
containers\nType=xfs\nOptions=defaults,prjquota\n\n[Install]\nRequiredBy=local-
fs.target","enabled":true,"name":"var-lib-containers.mount"}]}}'
[...]
```

検証

1. インストール中にまたはインストール後に、BareMetalHost オブジェクトがアノテーションを表示することを確認します。

```
$ oc get bmh -n my-sno-ns my-sno -ojson | jq '.metadata.annotations["bmac.agent-
install.openshift.io/ignition-config-overrides"]
```

出力例

```
"{\"ignition\":{\"version\":\"3.2.0\"},\"storage\":{\"disks\":[{\"device\":\"/dev/disk/by-
id/wwn-0x6b07b250ebb9d0002a33509f24af1f62\",\"partitions\":[{\"label\":\"var-lib-
containers\",\"sizeMiB\":0,\"startMiB\":250000}],\"wipeTable\":false}],\"filesystems\":
[{\"device\":\"/dev/disk/by-partlabel/var-lib-
containers\",\"format\":\"xfs\",\"mountOptions\":
[\"defaults\",\"prjquota\"],\"path\":\"/var/lib/containers\",\"wipeFilesystem\":true}]},\"sys
temd\":{\"units\":[{\"contents\":\"# Generated by Butane\\n[Unit]\\nRequires=systemd-
fsck@dev-disk-by\\\\x2dpartlabel-
```

```
var\\\x2dlib\\\x2dcontainers.service\\nAfter=systemd-fsck@dev-disk-
by\\\x2dpartlabel-
var\\\x2dlib\\\x2dcontainers.service\\n\\n[Mount]\\nWhere=/var/lib/containers\\nWhat=/
dev/disk/by-partlabel/var-lib-
containers\\nType=xfs\\nOptions=defaults,prjquota\\n\\n[Install]\\nRequiredBy=local-
fs.target\",\"enabled\":true,\"name\":\"var-lib-containers.mount\"}]}}"
```

2. インストール後に、単一ノードの OpenShift ディスクのステータスを確認します。

```
# lsblk
```

出力例

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 446.6G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0   127M  0 part
├─sda3   8:3    0   384M  0 part /boot
├─sda4   8:4    0 243.6G  0 part /var
│                       /sysroot/ostree/deploy/rhcos/var
│                       /usr
│                       /etc
│                       /
│                       /sysroot
└─sda5   8:5    0 202.5G  0 part /var/lib/containers
```

```
# df -h
```

出力例

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs           126G   84K  126G   1% /dev/shm
tmpfs            51G   93M   51G   1% /run
/dev/sda4       244G  5.2G  239G   3% /sysroot
tmpfs           126G  4.0K  126G   1% /tmp
/dev/sda5       203G  119G   85G  59% /var/lib/containers
/dev/sda3       350M  110M  218M  34% /boot
tmpfs            26G     0   26G   0% /run/user/1000
```

15.2.2. イメージベースのアップグレード用の Operator のインストール

Lifecycle Agent と OADP Operator をインストールして、クラスターをアップグレードする準備をします。

非 GitOps メソッドを使用して OADP Operator をインストールするには、OADP Operator のインストールを参照してください。

関連情報

- OADP Operator のインストール
- バックアップおよびスナップショットの場所、ならびにそのシークレットについて
- バックアップ CR の作成
- 復元 CR の作成

15.2.2.1. CLI を使用したライフサイクルエージェントのインストール

OpenShift CLI (oc)を使用して、Lifecycle Agent をインストールできます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. Lifecycle Agent の Namespace オブジェクト YAML ファイルを作成します（例： lcao-namespace.yaml ）。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-lifecycle-agent
  annotations:
    workload.openshift.io/allowed: management
```

a. 以下のコマンドを実行して Namespace CR を作成します。

```
$ oc create -f lcao-namespace.yaml
```

2. Lifecycle Agent の OperatorGroup オブジェクト YAML ファイルを作成します（例： lcao-operatorgroup.yaml ）。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-lifecycle-agent
  namespace: openshift-lifecycle-agent
spec:
  targetNamespaces:
  - openshift-lifecycle-agent
```

a. 以下のコマンドを実行して OperatorGroup CR を作成します。

```
$ oc create -f lcao-operatorgroup.yaml
```

3. Subscription CR （例： lcao-subscription.yaml ）を作成します。

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: openshift-lifecycle-agent-subscription
  namespace: openshift-lifecycle-agent
spec:
  channel: "stable"
  name: lifecycle-agent
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

a. 以下のコマンドを実行して Subscription CR を作成します。

```
$ oc create -f lcao-subscription.yaml
```

検証

1. インストールが成功したことを確認するには、次のコマンドを実行して CSV リソースを検査します。

```
$ oc get csv -n openshift-lifecycle-agent
```

出力例

```
NAME                         DISPLAY                VERSION           REPLACES   PHASE
lifecycle-agent.v4.16.0      Openshift Lifecycle Agent   4.16.0                Succeeded
```

2. 次のコマンドを実行して、ライフサイクルエージェントが稼働中であることを確認します。

```
$ oc get deploy -n openshift-lifecycle-agent
```

出力例

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
lifecycle-agent-controller-manager   1/1     1            1           14s
```

15.2.2.2. Web コンソールを使用したライフサイクルエージェントのインストール

OpenShift Container Platform Web コンソールを使用して Lifecycle Agent をインストールできます。

前提条件

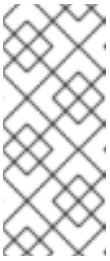
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Container Platform Web コンソールで、Operators → OperatorHub ページに移動します。
2. 利用可能な Operator の一覧から Lifecycle Agent を検索し、Install をクリックします。
3. Install Operator ページの A specific namespace on the cluster の下で openshift-ptp を選択します。
4. Install をクリックします。

検証

1. インストールが正常に行われたことを確認するには、以下を実行します。
   a. Operators → Installed Operators をクリックします。
   b. Status が InstallSucceeded の状態で、Lifecycle Agent が openshift-lifecycle-agent プロジェクトにリスト表示されていることを確認します。



注記

インストール時に、Operator は Failed ステータスを表示する可能性があります。インストールが後に InstallSucceeded メッセージを出して正常に実行される場合は、Failed メッセージを無視できます。

Operator が正常にインストールされていない場合、以下を実行します。

1. Operators → Installed Operators ページに移動し、Operator Subscriptions および Install Plans タブで Status にエラーがあるかどうかを検査します。

2. Workloads → Pods をクリックし、openshift-lifecycle-agent プロジェクトで Pod のログを確認します。

15.2.2.3. GitOps ZTP を使用したライフサイクルエージェントのインストール

GitOps Zero Touch Provisioning (ZTP)を使用して Lifecycle Agent をインストールして、イメージベースのアップグレードを行います。

前提条件

- source-crs ディレクトリーに custom-crs というディレクトリーを作成します。source-crs ディレクトリーは kustomization.yaml ファイルと同じ場所に存在する必要があります。

手順

1. 以下の CR を openshift-lifecycle-agent namespace に作成し、それらを source-crs/custom-crs ディレクトリーにプッシュします。

LcaSubscriptionNS.yaml ファイルの例

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-lifecycle-agent
  annotations:
    workload.openshift.io/allowed: management
    ran.openshift.io/ztp-deploy-wave: "2"
  labels:
    kubernetes.io/metadata.name: openshift-lifecycle-agent
```

LcaSubscriptionOperGroup.yaml ファイルの例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: lifecycle-agent-operatorgroup
  namespace: openshift-lifecycle-agent
```

```
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  targetNamespaces:
    - openshift-lifecycle-agent
```

LcaSubscription.yaml ファイルの例

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: lifecycle-agent
  namespace: openshift-lifecycle-agent
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: "stable"
  name: lifecycle-agent
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
  status:
    state: AtLatestKnown
```

ディレクトリー構造の例

```
├── kustomization.yaml
├── sno
│   ├── example-cnf.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   └── ns.yaml
├── source-crs
│   ├── custom-crs
│   │   ├── LcaSubscriptionNS.yaml
│   │   ├── LcaSubscriptionOperGroup.yaml
│   │   ├── LcaSubscription.yaml
```

2. 共通の PolicyGenTemplate に CR を追加します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true"
    du-profile: "latest"
  sourceFiles:
    - fileName: custom-crs/LcaSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/LcaSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/LcaSubscription.yaml
      policyName: "subscriptions-policy"
[...]
```

15.2.2.4. GitOps ZTP を使用した OADP Operator のインストールと設定

アップグレードを開始する前に、GitOps ZTP を使用して OADP Operator をインストールして設定します。

前提条件

- source-crs ディレクトリーに custom-crs というディレクトリーを作成します。source-crs ディレクトリーは kustomization.yaml ファイルと同じ場所に存在する必要があります。

手順

1. 以下の CR を openshift-adp namespace に作成し、それらを source-crs/custom-crs ディレクトリーにコピーします。

OadpSubscriptionNS.yaml ファイルの例

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-adp
  annotations:
```

```
    ran.openshift.io/ztp-deploy-wave: "2"
  labels:
    kubernetes.io/metadata.name: openshift-adp
```

OadpSubscriptionOperGroup.yaml ファイルの例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: redhat-oadp-operator
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  targetNamespaces:
  - openshift-adp
```

OadpSubscription.yaml ファイルの例

```
apiVersion: operators.coreos.com/v1
kind: Subscription
metadata:
  name: redhat-oadp-operator
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: stable-1.3
  name: redhat-oadp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
status:
  state: AtLatestKnown
```

OadpOperatorStatus.yaml ファイルの例

```
apiVersion: operators.coreos.com/v1
kind: Operator
metadata:
  name: redhat-oadp-operator.openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
status:
  components:
    refs:
    - kind: Subscription
      namespace: openshift-adp
      conditions:
      - type: CatalogSourcesUnhealthy
        status: "False"
    - kind: InstallPlan
      namespace: openshift-adp
      conditions:
      - type: Installed
        status: "True"
    - kind: ClusterServiceVersion
      namespace: openshift-adp
      conditions:
      - type: Succeeded
        status: "True"
        reason: InstallSucceeded
```

ディレクトリー構造の例

```
├── kustomization.yaml
├── sno
│   ├── example-cnf.yaml
│   ├── common-ranGen.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   └── ns.yaml
├── source-crs
│   ├── custom-crs
│   │   ├── OadpSubscriptionNS.yaml
│   │   ├── OadpSubscriptionOperGroup.yaml
│   │   ├── OadpSubscription.yaml
│   │   ├── OadpOperatorStatus.yaml
```

2. 共通の PolicyGenTemplate に CR を追加します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-common-latest"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true"
    du-profile: "latest"
  sourceFiles:
    - fileName: custom-crs/OadpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/OadpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/OadpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: custom-crs/OadpOperatorStatus.yaml
      policyName: "subscriptions-policy"
[...]
```

3. DataProtectionApplication CR と S3 シークレットを作成します。

   a. source-crs/custom-crs ディレクトリーに以下の CR を作成します。

DataProtectionApplication.yaml ファイルの例

```
apiVersion: oadp.openshift.io/v1
kind: DataProtectionApplication
metadata:
  name: dataprotectionapplication
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
spec:
  configuration:
    restic:
      enable: false (1)
    velero:
      defaultPlugins:
        - aws
        - openshift
      resourceTimeout: 10m
  backupLocations:
    - velero:
        config:
```

```
        profile: "default"
        region: minio
        s3Url: $url
        insecureSkipTLSVerify: "true"
        s3ForcePathStyle: "true"
      provider: aws
      default: true
      credential:
        key: cloud
        name: cloud-credentials
      objectStorage:
        bucket: $bucketName 2
        prefix: $prefixName 3
status:
  conditions:
  - reason: Complete
    status: "True"
    type: Reconciled
```

1

永続ボリュームの内容はアップグレード後に保持され、再利用されるため、イメージベースのアップグレードでは spec.configuration.restic.enable フィールドを false に設定する必要があります。

2 3

バケットは、S3 バックエンドで作成されるバケット名を定義します。接頭辞は、バケットに自動作成されるサブディレクトリーの名前を定義します。バケットと接頭辞の組み合わせは、クラスター間で干渉を回避するために、ターゲットクラスターごとに一意である必要があります。ターゲットクラスターごとに一意のストレージディレクトリーを確保するには、RHACM ハブテンプレート機能を使用できます（例 : prefix: {{hub .ManagedClusterName hub }}）。

OadpSecret.yaml ファイルの例

```
apiVersion: v1
kind: Secret
metadata:
  name: cloud-credentials
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
type: Opaque
```

OadpBackupStorageLocationStatus.yaml ファイルの例

```
apiVersion: velero.io/v1
kind: BackupStorageLocation
metadata:
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "100"
status:
  phase: Available
```

OadpBackupStorageLocationStatus.yaml CR は、OADP によって作成されたバックアップストレージロケーションの可用性を検証します。

b. オーバーライドを使用して、サイト PolicyGenTemplate に CR を追加します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "example-cnf"
  namespace: "ztp-site"
spec:
  bindingRules:
    sites: "example-cnf"
    du-profile: "latest"
  mcp: "master"
  sourceFiles:
    ...
    - fileName: custom-crs/OadpSecret.yaml
      policyName: "config-policy"
      data:
        cloud: <your_credentials> 1
    - fileName: custom-crs/DataProtectionApplication.yaml
      policyName: "config-policy"
      spec:
        backupLocations:
          - velero:
              config:
                region: minio
                s3Url: <your_S3_URL> 2
                profile: "default"
```

```
      insecureSkipTLSVerify: "true"
      s3ForcePathStyle: "true"
    provider: aws
    default: true
    credential:
      key: cloud
      name: cloud-credentials
    objectStorage:
      bucket: <your_bucket_name> 3
      prefix: <cluster_name> 4
- fileName: custom-crs/OadpBackupStorageLocationStatus.yaml
  policyName: "config-policy"
```

1

S3 ストレージバックエンドの認証情報を指定します。

2

S3 互換バケットの URL を指定します。

3 4

バケット は、S3 バックエンドで作成されるバケット名を定義します。接頭辞 は、バケット に自動作成されるサブディレクトリーの名前を定義します。バケット と 接頭辞 の組み合わせは、クラスター間で干渉を回避するために、ターゲットクラスターごとに一意である必要があります。ターゲットクラスターごとに一意のストレージディレクトリーを確保するには、RHACM ハブテンプレート機能を使用できます（例 : prefix: {{hub .ManagedClusterName hub }}）。

15.2.3. Lifecycle Agent を使用したイメージベースのアップグレード用のシードイメージの生成

Lifecycle Agent を使用して、Browse dGenerator カスタムリソース(CR)でシード イメージを生成します。

15.2.3.1. シードイメージ設定

シードイメージは、同様の設定を持つ単一ノード OpenShift クラスターのセットをターゲットにします。つまり、シードイメージには、シードクラスターがターゲットクラスターと共有するすべてのコンポーネントと設定が必要です。したがって、シードクラスターから生成されたシードイメージにはクラスター固有の設定を含めることはできません。

次の表に、シードイメージに含めなければならないコンポーネント、リソース、および設定を一覧表示しています。

表15.2 シードイメージ設定

Cluster configuration	シードイメージに含める
パフォーマンスプロファイル	はい
ターゲットクラスターの **MachineConfig** リソース	はい
ip_version = 4	はい
Lifecycle Agent および OADP Operator を含む Day 2 Operator の設定	はい
非接続レジストリーの設定	はい
有効なプロキシー設定 [2]	はい
FIPS 設定	はい
ターゲットクラスターのサイズと一致するコンテナーストレージ用のプライマリーディスク上の専用パーティション	はい
ローカルボリューム • LSO 向けに **LocalVolume** で使用される **StorageClass** • LSO の場合は **LocalVolume** • LVMS の **LVMCluster** CR	いいえ
OADP **DataProtectionApplication** CR	いいえ

1. **本リリースでは、デュアルスタックネットワークはサポート対象外です。**

2. **プロキシー設定は同じである必要はありません。**

15.2.3.1.1. RAN DU プロファイルを使用したシードイメージ設定

次の表に、RAN DU プロファイルの使用時にシードイメージに指定する必要のあるコンポーネント、リソース、および設定を一覧表示しています。

表15.3 RAN DU プロファイルを使用したシードイメージ設定

リソース	シードイメージに含める
Day 0 インストールの一部として適用されるすべての追加マニフェスト	はい
すべての Day 2 Operator サブスクリプション	はい
ClusterLogging.yaml	はい
DisableOLMPprof.yaml	はい
TunedPerformancePatch.yaml	はい
PerformanceProfile.yaml	はい
SriovOperatorConfig.yaml	はい
DisableSnoNetworkDiag.yaml	はい
StorageClass.yaml	**StorageLV.yaml**で使用される場合は No
StorageLV.yaml	いいえ
StorageLVMCluster.yaml	いいえ

表15.4 追加マニフェストのための RAN DU プロファイルを使用したシードイメージ設定

リソース	追加マニフェストとして適用
ClusterLogForwarder.yaml	はい
ReduceMonitoringFootprint.yaml	はい
SriovFecClusterConfig.yaml	はい
PtpOperatorConfigForEvent.yaml	はい
DefaultCatsrc.yaml	はい
PtpConfig.yaml	ターゲットクラスターのインターフェイスがシードクラスターで共通している場合は、シードイメージに追加できます。それ以外の場合は、これを追加のマニフェストとして適用します。

リソース	追加マニフェストとして適用
SriovNetwork.yamlSriovNetworkNodePolicy.yaml	namespace を含む設定がシードクラスターとターゲットクラスターの両方で完全に同じである場合は、それらをシードイメージに追加できます。それ以外の場合は、それらを追加のマニフェストとして適用します。

15.2.3.2. Lifecycle Agent を使用したシードイメージの生成

Lifecycle Agent を使用して、Browse dGenerator CR でシード イメージを生成します。Operator は必要なシステム設定の有無を確認し、シードイメージを生成する前に必要なシステムクリーンアップを実行し、イメージ生成を起動します。seed イメージの生成には以下のタスクが含まれます。

- **クラスター Operator の停止**
- **シードイメージ設定の準備**
- **SeedGenerator CR で指定されたイメージリポジトリーにシードイメージを生成およびプッシュする**
- **クラスター Operator の復元**
- **シードクラスター証明書の有効期限**
- **シードクラスターの新しい証明書の生成**
- **シードクラスターの SeedGenerator CR の復元と更新**

前提条件

- **シードクラスターに共有コンテナーディレクトリーを設定します。**

- OADP Operator と Lifecycle Agent をシードクラスターにインストールします。

手順

1. ハブからクラスターをデタッチして、シードイメージにしてはならないシードクラスターからクラスター固有のリソースを削除します。

   a. RHACM を使用している場合は、次のコマンドを実行して、シードクラスターを手動で切り離します。

   ```
   $ oc delete managedcluster sno-worker-example
   ```

      i. ManagedCluster CR が削除されるまで待ちます。CR が削除されたら、適切な SeedGenerator CR を作成します。Lifecycle Agent は RHACM アーティファクトをクリーンアップします。

   b. GitOps ZTP を使用している場合は、kustomization.yaml からシードクラスターの SiteConfig CR を削除して、クラスターをデタッチします。

      i. kustomization.yaml からシードクラスターの SiteConfig CR を削除します。

      ```
      apiVersion: kustomize.config.k8s.io/v1beta1
      kind: Kustomization

      generators:
      #- example-seed-sno1.yaml
      - example-target-sno2.yaml
      - example-target-sno3.yaml
      ```

      ii. kustomization.yaml の変更を Git リポジトリーにコミットし、変更をプッシュします。

      ArgoCD パイプラインは変更を検出し、マネージドクラスターを削除します。

2. シークレットを作成します。

   a. 次のコマンドを実行して、認証ファイルを作成します。

```
$ MY_USER=myuserid
$ AUTHFILE=/tmp/my-auth.json
$ podman login --authfile ${AUTHFILE} -u ${MY_USER} quay.io/${MY_USER}
```

```
$ base64 -w 0 ${AUTHFILE} ; echo
```

b. 出力を openshift-lifecycle-agent namespace の seedgen という名前の Secret YAML ファイルの seedAuth フィールドにコピーします。

```
apiVersion: v1
kind: Secret
metadata:
  name: seedgen 1
  namespace: openshift-lifecycle-agent
type: Opaque
data:
  seedAuth: <encoded_AUTHFILE> 2
```

1 Secret リソースの name: seedgen および namespace: openshift-lifecycle-agent フィールドが必要です。

2 生成されたシードイメージをプッシュするためにレジストリーへの書き込みアクセス用に base64 でエンコードされた authfile を指定します。

c. 以下のコマンドを実行してシークレットを作成します。

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

3. SeedGenerator CR を作成します。

```
apiVersion: lca.openshift.io/v1
kind: SeedGenerator
metadata:
  name: seedimage 1
spec:
  seedImage: <seed_container_image> 2
```

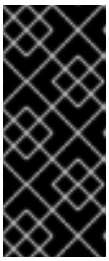
1 SeedGenerator CR は seedimage という名前にする必要があります。

2

コンテナーイメージの URL を指定します（例： quay.io/example/seed-container-image:<tag>）。< seed_cluster_name>:<ocp_version> 形式を使用することが推奨されます。

4. 次のコマンドを実行して、シードイメージを生成します。

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



重要

Lifecycle Agent がシードイメージを生成する間、クラスターは再起動し、API 機能が失われます。SeedGenerator CR を適用すると、kubelet と CRI-O 操作が停止し、イメージの生成が開始されます。

さらにシードイメージを生成する場合は、シードイメージを生成するバージョンで新しいシードクラスターをプロビジョニングする必要があります。

検証

1. クラスターが復旧し、使用可能になったら、次のコマンドを実行して SeedGenerator CR のステータスを確認できます。

```
$ oc get seedgenerator -o yaml
```

出力例

```
status:
  conditions:
  - lastTransitionTime: "2024-02-13T21:24:26Z"
    message: Seed Generation completed
    observedGeneration: 1
    reason: Completed
    status: "False"
    type: SeedGenInProgress
  - lastTransitionTime: "2024-02-13T21:24:26Z"
    message: Seed Generation completed
    observedGeneration: 1
    reason: Completed
```

```
    status: "True"
    type: SeedGenCompleted 1
  observedGeneration: 1
```

1 シードイメージの生成が完了しました。

関連情報

- ostree stateroots 間で共有コンテナーディレクトリーの設定
- GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設定

15.2.4. Lifecycle Agent を使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成

ライフサイクルエージェントは、イメージベースのアップグレード用にそれらを処理するために、すべての OADP リソース、追加のマニフェスト、および ConfigMap オブジェクトでラップされたカスタムカタログソースを必要とします。

15.2.4.1. ライフサイクルエージェントを使用したイメージベースのアップグレード用の OADP ConfigMap オブジェクトの作成

アップグレード中にリソースをバックアップおよび復元するために使用される OADP リソースを作成します。

前提条件

- 互換性のあるシードクラスターからシードイメージを生成します。
- OADP バックアップおよび復元リソースを作成します。
- stateroot 間で共有されるコンテナーイメージ用に、ターゲットクラスターに別のパーティションを作成します。詳細については、イメージベースのアップグレード用の共有コンテ

ナーディレクトリーの設定を参照してください。

- シードイメージで使用されるバージョンと互換性のあるバージョンの Lifecycle Agent をデプロイします。

- OADP Operator、DataProtectionApplication CR、およびそのシークレットをターゲットクラスターにインストールします。

- S3 互換ストレージソリューションと、適切な認証情報が設定されたすぐに使用できるバケットを作成します。詳細は、About installing OADP を参照してください。

手順

1. OADP Operator がインストールされているのと同じ namespace (openshift-adp)に、プラットフォームアーティファクトの OADP バックアップ CR および Restore CR を作成します。

   a. ターゲットクラスターが RHACM によって管理される場合は、RHACM アーティファクトをバックアップおよび復元するために次の YAML ファイルを追加します。

   RHACM の PlatformBackupRestore.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  annotations:
    lca.openshift.io/apply-label: "apps/v1/deployments/open-cluster-management-agent/klusterlet,v1/secrets/open-cluster-management-agent/bootstrap-hub-kubeconfig,rbac.authorization.k8s.io/v1/clusterroles/klusterlet,v1/serviceaccounts/open-cluster-management-agent/klusterlet,scheduling.k8s.io/v1/priorityclasses/klusterlet-critical,rbac.authorization.k8s.io/v1/clusterroles/open-cluster-management:klusterlet-admin-aggregate-clusterrole,rbac.authorization.k8s.io/v1/clusterrolebindings/klusterlet,operator.open-cluster-management.io/v1/klusterlets/klusterlet,apiextensions.k8s.io/v1/customresourcedefinitions/klusterlets.operator.open-cluster-management.io,v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials" 1
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
```

```
spec:
  includedNamespaces:
  - open-cluster-management-agent
  includedClusterScopedResources:
  - klusterlets.operator.open-cluster-management.io
  - clusterroles.rbac.authorization.k8s.io
  - clusterrolebindings.rbac.authorization.k8s.io
  - priorityclasses.scheduling.k8s.io
  includedNamespaceScopedResources:
  - deployments
  - serviceaccounts
  - secrets
  excludedNamespaceScopedResources: []
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "1"
spec:
  backupName:
    acm-klusterlet
```

1

multiclusterHub CR に .spec.imagePullSecret が定義されておらず、シークレットがハブクラスターの open-cluster-management-agent namespace に存在しない場合は、v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials を削除します。

b. LVM ストレージを使用してクラスターに永続ボリュームを作成した場合は、LVM ストレージアーティファクトに以下の YAML ファイルを追加します。

PlatformBackupRestoreLvms.yaml for LVM Storage

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: lvmcluster
```

```
  namespace: openshift-adp
spec:
  includedNamespaces:
    - openshift-storage
  includedNamespaceScopedResources:
    - lvmclusters
    - lvmvolumegroups
    - lvmvolumegroupnodestatuses
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: lvmcluster
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "2" (1)
spec:
  backupName:
    lvmcluster
```

(1) lca.openshift.io/apply-wave 値は、アプリケーションの Restore CR で指定された値よりも低い値である必要があります。

2. （オプション）アップグレード後にアプリケーションを復元する必要がある場合は、openshift-adp namespace にアプリケーションの OADP Backup および Restore CR を作成します。

   a. クラスタースコープのアプリケーションアーティファクトの OADP CR を openshift-adp namespace に作成します。

   LSO および LVM ストレージ用のクラスタースコープのアプリケーションアーティファクトの OADP CR の例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    lca.openshift.io/apply-label:
"apiextensions.k8s.io/v1/customresourcedefinitions/test.example.com,security.op
```

```
enshift.io/v1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterrol
es/test-
role,rbac.authorization.k8s.io/v1/clusterrolebindings/system:openshift:scc:test"
1
  name: backup-app-cluster-resources
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedClusterScopedResources:
  - customresourcedefinitions
  - securitycontextconstraints
  - clusterrolebindings
  - clusterroles
  excludedClusterScopedResources:
  - Namespace
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app-cluster-resources
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3" 2
spec:
  backupName:
    backup-app-cluster-resources
```

1 サンプルリソース名は、実際のリソースに置き換えます。

2 lca.openshift.io/apply-wave 値は、プラットフォームの Restore CR の値よりも高く、アプリケーションの namespace スコープの Restore CR の値よりも低い値である必要があります。

b. namespace スコープのアプリケーションアーティファクトの OADP CR を作成します。

LSO が使用されている場合の OADP CR namespace スコープのアプリケーションアーティファクトの例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  excludedClusterScopedResources:
  - persistentVolumes
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
```

1

アプリケーションのカスタムリソースを定義します。

LVM ストレージが使用されている場合の OADP CR namespace スコープのアプリケーションアーティファクトの例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  includedClusterScopedResources:
  - persistentVolumes 2
  - logicalvolumes.topolvm.io 3
  - volumesnapshotcontents 4
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
  restorePVs: true
  restoreStatus:
    includedResources:
    - logicalvolumes 5
```

1

アプリケーションのカスタムリソースを定義します。

2

必須フィールド。

3

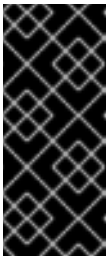
必須フィールド。

4

LVM ストレージボリュームスナップショットを使用する場合はオプション。

5

必須フィールド。



重要

アプリケーションの同じバージョンは、OpenShift Container Platform の現行リリースとターゲットリリースの両方で機能する必要があります。

3. 次のコマンドを実行して、OADP CR の ConfigMap オブジェクトを作成します。

```
$ oc create configmap oadp-cm-example --from-file=example-oadp-resources.yaml=
<path_to_oadp_crs> -n openshift-adp
```

4. 次のコマンドを実行して、ImageBasedUpgrade CR にパッチを適用します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"oadpContent": [{"name": "oadp-cm-example", "namespace":
"openshift-adp"}]}}' \
  --type=merge -n openshift-lifecycle-agent
```

関連情報

- ostree stateroots 間で共有コンテナーディレクトリーの設定
- OADP のインストールについて

15.2.4.2. Lifecycle Agent を使用したイメージベースのアップグレードの追加マニフェストの ConfigMap オブジェクトの作成

ターゲットクラスターに適用する追加のマニフェストを作成できます。

手順

1. SR-IOV などの追加のマニフェストを含む YAML ファイルを作成します。

SR-IOV リソースの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: "example-sriov-node-policy"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  isRdma: false
  nicSelector:
    pfNames: [ens1f0]
  nodeSelector:
    node-role.kubernetes.io/master: ""
  mtu: 1500
  numVfs: 8
  priority: 99
  resourceName: example-sriov-node-policy
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: "example-sriov-network"
  namespace: openshift-sriov-network-operator
spec:
  ipam: |-
    {
    }
  linkState: auto
  networkNamespace: sriov-namespace
  resourceName: example-sriov-node-policy
  spoofChk: "on"
  trust: "off"
```

2. 以下のコマンドを実行して設定マップオブジェクトを作成します。

```
$ oc create configmap example-extra-manifests-cm --from-file=example-extra-
manifests.yaml=<path_to_extramanifest> -n openshift-lifecycle-agent
```

3. 次のコマンドを実行して、ImageBasedUpgrade CR にパッチを適用します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"extraManifests": [{"name": "example-extra-manifests-cm",
"namespace": "openshift-lifecycle-agent"}]}}' \
  --type=merge -n openshift-lifecycle-agent
```

15.2.4.3. Lifecycle Agent を使用したイメージベースのアップグレードのカスタムカタログソースの ConfigMap オブジェクトの作成

カタログソースの ConfigMap オブジェクトを生成し、それらを ImageBasedUpgrade CR の spec.extraManifest フィールドに追加することで、アップグレード後にカスタムカタログソースを維持できます。カタログソースの詳細は、カタログソースを参照してください。

手順

1. CatalogSource CR を含む YAML ファイルを作成します。

```
apiVersion: operators.coreos.com/v1
kind: CatalogSource
metadata:
  name: example-catalogsources
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  displayName: disconnected-redhat-operators
  image: quay.io/example-org/example-catalog:v1
```

2. 以下のコマンドを実行して設定マップオブジェクトを作成します。

```
$ oc create configmap example-catalogsources-cm --from-file=example-
catalogsources.yaml=<path_to_catalogsource_cr> -n openshift-lifecycle-agent
```

3. 次のコマンドを実行して、ImageBasedUpgrade CR にパッチを適用します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade \
  -p='{"spec": {"extraManifests": [{"name": "example-catalogsources-cm",
```

```
"namespace": "openshift-lifecycle-agent"}]}}' \
  --type=merge -n openshift-lifecycle-agent
```

関連情報

- カタログソース
- Lifecycle Agent を使用したイメージベースのアップグレードの実行

15.2.5. GitOps ZTP を使用したライフサイクルエージェントによるイメージベースのアップグレード用の ConfigMap オブジェクトの作成

OADP リソース、追加のマニフェスト、およびカスタムカタログソースを ConfigMap オブジェクトにラップして、イメージベースのアップグレードの準備をします。

15.2.5.1. GitOps ZTP を使用したイメージベースのアップグレード用の OADP リソースの作成

アップグレード後にアプリケーションを復元するために、OADP リソースを準備します。

前提条件

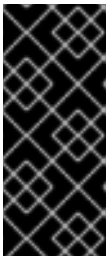
- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- 互換性のあるシードクラスターからシードイメージを生成します。
- stateroot 間で共有されるコンテナーイメージ用に、ターゲットクラスターに別のパーティションを作成します。詳細は、「GitOps ZTP 使用時の ostree stateroots 間で共有コンテナーディレクトリーの設定」を参照してください。
- シードイメージで使用されるバージョンと互換性のあるバージョンの Lifecycle Agent をデプロイします。
- OADP Operator、DataProtectionApplication CR、およびそのシークレットをターゲットクラスターにインストールします。

- S3 互換ストレージソリューションと、適切な認証情報が設定されたすぐに使用できるバケットを作成します。詳細については、「GitOps ZTP を使用した OADP Operator のインストールと設定」を参照してください。

手順

1. ArgoCD policies アプリケーションで使用する Git リポジトリーに以下のディレクトリー構造が含まれていることを確認します。

```
├── source-crs/
│   ├── ibu/
│   │   ├── ImageBasedUpgrade.yaml
│   │   ├── PlatformBackupRestore.yaml
│   │   ├── PlatformBackupRestoreLvms.yaml
├── ...
├── ibu-upgrade-ranGen.yaml
├── kustomization.yaml
```



重要

kustomization.yaml ファイルは、ibu-upgrade-ranGen.yaml マニフェストを参照するために以前に示したディレクトリー構造と同じディレクトリー構造に存在する必要があります。

source-crs/ibu/PlatformBackupRestore.yaml ファイルは ZTP コンテナーイメージで提供されます。

PlatformBackupRestore.yaml

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: acm-klusterlet
  annotations:
    lca.openshift.io/apply-label: "apps/v1/deployments/open-cluster-management-agent/klusterlet,v1/secrets/open-cluster-management-agent/bootstrap-hub-kubeconfig,rbac.authorization.k8s.io/v1/clusterroles/klusterlet,v1/serviceaccounts/open-cluster-management-agent/klusterlet,scheduling.k8s.io/v1/priorityclasses/klusterlet-critical,rbac.authorization.k8s.io/v1/clusterroles/open-cluster-management:klusterlet-admin-aggregate-clusterrole,rbac.authorization.k8s.io/v1/clusterrolebindings/klusterlet,operator.open-cluster-
```

```
management.io/v1/klusterlets/klusterlet,apiextensions.k8s.io/v1/customresourcedefinit
ions/klusterlets.operator.open-cluster-management.io,v1/secrets/open-cluster-
management-agent/open-cluster-management-image-pull-credentials" 1
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedNamespaces:
  - open-cluster-management-agent
  includedClusterScopedResources:
  - klusterlets.operator.open-cluster-management.io
  - clusterroles.rbac.authorization.k8s.io
  - clusterrolebindings.rbac.authorization.k8s.io
  - priorityclasses.scheduling.k8s.io
  includedNamespaceScopedResources:
  - deployments
  - serviceaccounts
  - secrets
  excludedNamespaceScopedResources: []
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: acm-klusterlet
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "1"
spec:
  backupName:
    acm-klusterlet
```

1

multiclusterHub CR に .spec.imagePullSecret が定義されておらず、シークレットがハブクラスターの open-cluster-management-agent namespace に存在しない場合は、v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials を削除します。

LVM ストレージを使用して永続ボリュームを作成する場合は、ZTP コンテナーイメージで提供される source-crs/ibu/PlatformBackupRestoreLvms.yaml を使用して LVM ストレージリソースをバックアップできます。

PlatformBackupRestoreLvms.yaml

-

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: lvmcluster
  namespace: openshift-adp
spec:
  includedNamespaces:
    - openshift-storage
  includedNamespaceScopedResources:
    - lvmclusters
    - lvmvolumegroups
    - lvmvolumegroupnodestatuses
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: lvmcluster
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "2" (1)
spec:
  backupName:
    lvmcluster
```

(1) lca.openshift.io/apply-wave 値は、アプリケーションの Restore CR で指定された値よりも低い値である必要があります。

2. オプション：アップグレード後にアプリケーションを復元する必要がある場合は、openshift-adp namespace にアプリケーションの OADP Backup CR および Restore CR を作成します。

   a. クラスタースコープのアプリケーションアーティファクトの OADP CR を openshift-adp namespace に作成します。

   LSO および LVM ストレージ用のクラスタースコープのアプリケーションアーティファクトの OADP CR の例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    lca.openshift.io/apply-label:
"apiextensions.k8s.io/v1/customresourcedefinitions/test.example.com,security.op
enshift.io/v1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterrol
es/test-
role,rbac.authorization.k8s.io/v1/clusterrolebindings/system:openshift:scc:test"
(1)
  name: backup-app-cluster-resources
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  includedClusterScopedResources:
  - customresourcedefinitions
  - securitycontextconstraints
  - clusterrolebindings
  - clusterroles
  excludedClusterScopedResources:
  - Namespace
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app-cluster-resources
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3" (2)
spec:
  backupName:
    backup-app-cluster-resources
```

(1) サンプルリソース名は、実際のリソースに置き換えます。

(2) lca.openshift.io/apply-wave 値は、プラットフォームの Restore CR の値よりも高く、アプリケーションの namespace スコープの Restore CR の値よりも低い値である必要があります。

b. namespace スコープのアプリケーションアーティファクトの OADP CR を source-crs/custom-crs ディレクトリーに作成します。

LSO が使用されている場合の OADP CR namespace スコープのアプリケーションアーティファクトの例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  excludedClusterScopedResources:
  - persistentVolumes
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
```

1

アプリケーションのカスタムリソースを定義します。

LVM ストレージが使用されている場合の OADP CR namespace スコープのアプリケーションアーティファクトの例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: backup-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  - configmaps
  - cronjobs
  - services
  - job
  - poddisruptionbudgets
  - <application_custom_resources> 1
  includedClusterScopedResources:
  - persistentVolumes 2
  - logicalvolumes.topolvm.io 3
  - volumesnapshotcontents 4
---
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: test-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "4"
spec:
  backupName:
    backup-app
  restorePVs: true
  restoreStatus:
    includedResources:
    - logicalvolumes 5
```

1

アプリケーションのカスタムリソースを定義します。

2

必須フィールド。

3

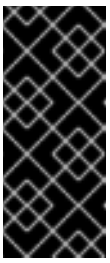
必須フィールド。

4

LVM ストレージボリュームスナップショットを使用する場合はオプション。

5

必須フィールド。



重要

アプリケーションの同じバージョンは、OpenShift Container Platform の現行リリースとターゲットリリースの両方で機能する必要があります。

3. ibu-upgrade-ranGen.yaml という新しい PolicyGenTemplate に、 oadp-cm -policy を介して oadp-cm ConfigMap オブジェクトを作成します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: example-group-ibu
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  evaluationInterval:
    compliant: 10s
    noncompliant: 10s
  sourceFiles:
  - fileName: ConfigMapGeneric.yaml
    complianceType: mustonlyhave
    policyName: "oadp-cm-policy"
```

```
  metadata:
    name: oadp-cm
    namespace: openshift-adp
```

4. 以下の内容で kustomization.yaml を作成します。

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators: (1)
- ibu-upgrade-ranGen.yaml

configMapGenerator: (2)
- files:
  - source-crs/ibu/PlatformBackupRestore.yaml
  #- source-crs/custom-crs/ApplicationClusterScopedBackupRestore.yaml
  #- source-crs/custom-crs/ApplicationApplicationBackupRestoreLso.yaml
  name: oadp-cm
  namespace: ztp-group
generatorOptions:
  disableNameSuffixHash: true


patches: (3)
- target:
    group: policy.open-cluster-management.io
    version: v1
    kind: Policy
    name: group-ibu-oadp-cm-policy
  patch: |-
    - op: replace
      path: /spec/policy-templates/0/objectDefinition/spec/object-
templates/0/objectDefinition/data
      value: '{{hub copyConfigMapData "ztp-group" "oadp-cm" hub}}'
```

(1) oadp-cm-policy を生成します。

(2) Backup CR と Restore CR を使用して、ハブクラスターに oadp-cm ConfigMap オブジェクトを作成します。

(3) oadp-cm-policy に追加された ConfigMap の data フィールドをオーバーライドします。ハブテンプレートは、oadp-cm ConfigMap をすべてのターゲットクラスターに伝播するために使用されます。

5. 変更を Git リポジトリーにプッシュします。

関連情報

- GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設定
- GitOps ZTP を使用した OADP Operator のインストールと設定

15.2.5.2. GitOps ZTP を使用したイメージベースのアップグレードの追加マニフェストのラベル付け

Lifecycle Agent が lca.openshift.io/target-ocp-version: <target_version> ラベルのラベルが付けられたリソースを抽出できるように、追加のマニフェストにラベルを 付けます。

前提条件

- GitOps ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- 互換性のあるシードクラスターからシードイメージを生成します。
- stateroot 間で共有されるコンテナーイメージ用に、ターゲットクラスターに別のパーティションを作成します。詳細は、「GitOps ZTP 使用時の ostree stateroots 間で共有コンテナーディレクトリーの設定」を参照してください。
- シードイメージで使用されるバージョンと互換性のあるバージョンの Lifecycle Agent をデプロイします。

手順

1. 必要な追加マニフェストに、既存の PolicyGenTemplate CR の lca.openshift.io/target-ocp-version: <target_version > ラベルでラベルを付けます。

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

```
kind: PolicyGenTemplate
metadata:
  name: example-sno
spec:
  bindingRules:
    sites: "example-sno"
    du-profile: "4.15"
  mcp: "master"
  sourceFiles:
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-fh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15" (1)
      spec:
        resourceName: du_fh
        vlan: 140
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-fh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15"
      spec:
        deviceType: netdevice
        isRdma: false
        nicSelector:
          pfNames: ["ens5f0"]
        numVfs: 8
        priority: 10
        resourceName: du_fh
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-mh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15"
      spec:
        resourceName: du_mh
        vlan: 150
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-mh"
        labels:
          lca.openshift.io/target-ocp-version: "4.15"
      spec:
        deviceType: vfio-pci
        isRdma: false
        nicSelector:
          pfNames: ["ens7f0"]
        numVfs: 8
        priority: 10
        resourceName: du_mh
    - fileName: DefaultCatsrc.yaml (2)
```

```
  policyName: "config-policy"
  metadata:
    name: default-cat-source
    namespace: openshift-marketplace
    labels:
      lca.openshift.io/target-ocp-version: "4.15"
  spec:
    displayName: default-cat-source
    image: quay.io/example-org/example-catalog:v1
```

1 lca.openshift.io/target-ocp-version ラベルが、ImageBasedUpgrade CR の spec.seedImageRef.version フィールドで指定されているターゲット OpenShift Container Platform バージョンの y-stream または z-stream のいずれかと一致していることを確認します。Lifecycle Agent は、指定されたバージョンに一致する CR のみを適用します。

2 カスタムカタログソースを使用しない場合は、このエントリーを削除します。

2. 変更を Git リポジトリーにプッシュします。

関連情報

- GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設定
- GitOps ZTP を使用した単一ノード OpenShift クラスターのイメージベースのアップグレードの実行

15.3. 単一ノードの OPENSHIFT クラスターのイメージベースのアップグレードの実行

Lifecycle Agent を使用して、シングルノード OpenShift クラスターのイメージベースの手動アップグレードを実行できます。

Lifecycle Agent をクラスターにデプロイすると、ImageBasedUpgrade CR が自動的に作成されます。この CR を更新して、シードイメージのイメージリポジトリーを指定し、別のステージを通過します。

15.3.1. Lifecycle Agent を使用したイメージベースのアップグレードの Prep ステージへの移行

Lifecycle Agent をクラスターにデプロイすると、ImageBasedUpgrade カスタムリソース(CR)が自動的に作成されます。

アップグレード中に必要なすべてのリソースを作成した後、Prep 段階に進むことができます。詳細は、ライフサイクルエージェントを使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成セクションを参照してください。

前提条件

- クラスターのバックアップおよび復元用のリソースを作成している。

手順

1. ImageBasedUpgrade CR にパッチを適用していることを確認します。

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle
  seedImageRef:
    version: 4.15.2 (1)
    image: <seed_container_image> (2)
    pullSecretRef: <seed_pull_secret> (3)
  autoRollbackOnFailure: {}
#   initMonitorTimeoutSeconds: 1800 (4)
  extraManifests: (5)
  - name: example-extra-manifests-cm
    namespace: openshift-lifecycle-agent
  - name: example-catalogsources-cm
    namespace: openshift-lifecycle-agent
  oadpContent: (6)
  - name: oadp-cm-example
    namespace: openshift-adp
```

(1) ターゲットプラットフォームのバージョンを指定します。この値は、シードイメージのバージョンと一致する必要があります。

(2) ターゲットクラスターがシードイメージをプルできるリポジトリーを指定します。

3 イメージがプライベートレジストリーにある場合は、コンテナーイメージをプルするための認証情報でシークレットへの参照を指定します。

4 （オプション）最初の再起動後にアップグレードが完了しない場合に、ロールバックする時間枠を秒単位で指定します。定義されていないか、0 に設定されている場合は、デフォルト値の 1800 秒(30 分)が使用されます。

5 （オプション）アップグレード後に保持するカスタムカタログソースを含む ConfigMap リソースの一覧と、シードイメージの一部ではないターゲットクラスターに適用する追加のマニフェストを指定します。

6 OADP ConfigMap 情報とともに oadpContent セクションを追加します。

2. Prep ステージを開始するには、次のコマンドを実行して、ImageBasedUpgrade CR の stage フィールドの値を Prep に変更します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Prep"}}' --type=merge -n openshift-lifecycle-agent
```

OADP リソースに ConfigMap オブジェクトと追加のマニフェストを指定した場合、ライフサイクルエージェントは、Prep の段階で指定された ConfigMap オブジェクトを検証します。以下の問題が発生する可能性があります。

- Lifecycle エージェントが extraManifests パラメーターの問題を検出すると、検証の警告またはエラー。
- oadpContent パラメーターに関する問題をライフサイクルエージェントが検出した場合の検証エラー。

検証の警告は Upgrade ステージをブロックしませんが、アップグレードを続行しても安全かどうかを判断する必要があります。CRD、namespace、またはドライランの失敗の欠落など、これらの警告は、ImageBasedUpgrade CR の Prep stage と annotation フィールドの status.conditions を警告の詳細で更新します。

検証警告の例

```
[...]
metadata:
annotations:
  extra-manifest.lca.openshift.io/validation-warning: '...'
[...]
```

ただし、MachineConfig または Operator マニフェストを追加マニフェストに追加するなど、検証エラーにより、Prep 段階が失敗し、アップグレード ステージがブロックされます。

検証にパスすると、クラスターは新規の ostree stateroot を作成します。これには、シードイメージをプルおよび展開し、ホストレベルのコマンドを実行します。最後に、必要なすべてのイメージがターゲットクラスターに事前キャッシュされます。

検証

- 次のコマンドを実行して、ImageBasedUpgrade CR のステータスを確認します。

```
$ oc get ibu -o yaml
```

出力例

```
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 13
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 13
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep stage completed successfully
    observedGeneration: 13
```

```
    reason: Completed
    status: "True"
    type: PrepCompleted
  observedGeneration: 13
  validNextStages:
  - Idle
  - Upgrade
```

関連情報

- Lifecycle Agent を使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成

15.3.2. Lifecycle Agent を使用したイメージベースのアップグレードのアップグレードステージへの移行

シードイメージを生成して Prep 段階を完了したら、ターゲットクラスターをアップグレードできます。アップグレードプロセス中に、OADP Operator は OADP カスタムリソース(CR)で指定されたアーティファクトのバックアップを作成し、ライフサイクルエージェントがクラスターをアップグレードします。

アップグレードが失敗するか停止すると、自動ロールバックが開始されます。アップグレード後に問題が発生した場合は、手動のロールバックを開始できます。手動ロールバックの詳細は、（オプション）Lifecycle Agent を使用したロールバックの開始 を参照してください。

前提条件

- Prep ステージを完了します。

手順

1. Upgrade stage に移行するには、次のコマンドを実行して、ImageBased Upgrade CR の stage フィールドの値を Upgrade に変更します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Upgrade"}}' --type=merge
```

2. 次のコマンドを実行して、ImageBasedUpgrade CR のステータスを確認します。

```
$ oc get ibu -o yaml
```

出力例

```
status:
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 5
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 5
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed successfully
    observedGeneration: 5
    reason: Completed
    status: "True"
    type: PrepCompleted
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: |-
      Waiting for system to stabilize: one or more health checks failed
        - one or more ClusterOperators not yet ready: authentication
        - one or more MachineConfigPools not yet ready: master
        - one or more ClusterServiceVersions not yet ready: sriov-fec.v2.8.0
    observedGeneration: 1
    reason: InProgress
    status: "True"
    type: UpgradeInProgress
  observedGeneration: 1
  rollbackAvailabilityExpiration: "2024-05-19T14:01:52Z"
  validNextStages:
  - Rollback
```

OADP Operator は、OADP Backup CR と Restore CR で指定されたデータのバックアップを作成し、ターゲットクラスターを再起動します。

3. 次のコマンドを実行して、CR のステータスを監視します。

```
$ oc get ibu -o yaml
```

4. アップグレードが適切であれば、次のコマンドを実行して、ImageBasedUpgrade CR の stage フィールドの値を Idle にパッチを適用して、変更を完了します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Idle"}}' --type=merge
```

重要

アップグレード後に Idle ステージに移動した後、変更をロールバックすることはできません。

ライフサイクルエージェントは、アップグレードプロセスで作成されたすべてのリソースを削除します。

検証

1. 次のコマンドを実行して、ImageBasedUpgrade CR のステータスを確認します。

```
$ oc get ibu -o yaml
```

出力例

```
status:
  conditions:
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: In progress
    observedGeneration: 5
    reason: InProgress
    status: "False"
    type: Idle
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed
    observedGeneration: 5
    reason: Completed
    status: "False"
    type: PrepInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Prep completed successfully
    observedGeneration: 5
    reason: Completed
    status: "True"
```

```
    type: PrepCompleted
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Upgrade completed
    observedGeneration: 1
    reason: Completed
    status: "False"
    type: UpgradeInProgress
  - lastTransitionTime: "2024-01-01T09:00:00Z"
    message: Upgrade completed
    observedGeneration: 1
    reason: Completed
    status: "True"
    type: UpgradeCompleted
  observedGeneration: 1
  rollbackAvailabilityExpiration: "2024-01-01T09:00:00Z"
  validNextStages:
  - Idle
  - Rollback
```

2. 次のコマンドを実行して、クラスター復元のステータスを確認します。

```
$ oc get restores -n openshift-adp -o custom-
columns=NAME:.metadata.name,Status:.status.phase,Reason:.status.failureReason
```

出力例

```
NAME           Status      Reason
acm-klusterlet   Completed   <none> 1
apache-app       Completed   <none>
localvolume      Completed   <none>
```

1. acm-klusterlet は、RHACM 環境のみに固有のものです。

関連情報

- （オプション）Lifecycle Agent を使用したイメージベースのアップグレードのロールバックステージへの移行

15.3.3. （オプション）Lifecycle Agent を使用したイメージベースのアップグレードのロールバックステージへの移行

再起動後に initMonitorTimeoutSeconds フィールドに指定された時間枠内にアップグレードが完了しない場合に、自動ロールバックが開始されます。

ImageBasedUpgrade CR の例

```
apiVersion: lca.openshift.io/v1
kind: ImageBasedUpgrade
metadata:
  name: upgrade
spec:
  stage: Idle
  seedImageRef:
    version: 4.15.2
    image: <seed_container_image>
  autoRollbackOnFailure: {}
#    initMonitorTimeoutSeconds: 1800 (1)
[...]
```

(1) （オプション）最初の再起動後にアップグレードが完了しない場合に、ロールバックする時間枠を秒単位で指定します。定義されていないか、0 に設定されている場合は、デフォルト値の 1800 秒(30 分)が使用されます。

アップグレード後に解決不可能な問題が発生した場合は、変更を手動でロールバックできます。

前提条件

- cluster-admin 権限を持つユーザーとしてクラスターにログインしている。

手順

1. ロールバックステージに移行するには、次のコマンドを実行して、ImageBasedUpgrade CR で stage フィールドの値を Rollback にパッチを適用します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Rollback"}}' --type=merge
```

Lifecycle Agent は、以前にインストールされたバージョンの OpenShift Container Platform でクラスターを再起動し、アプリケーションを復元します。

2. 変更に問題がなければ、次のコマンドを実行して、stage フィールドの値を ImageBasedUpgrade CR の Idle にパッチを適用してロールバックを完了します。

```
$ oc patch imagebasedupgrades.lca.openshift.io upgrade -p='{"spec": {"stage":
"Idle"}}' --type=merge -n openshift-lifecycle-agent
```

警告

ロールバック後に Idle 段階に移行すると、ライフサイクルエージェントは、失敗したアップグレードのトラブルシューティングに使用できるリソースをクリーンアップします。

15.3.4. Lifecycle Agent を使用したイメージベースのアップグレードのトラブルシューティング

イメージベースのアップグレード中に問題が発生する可能性があります。

15.3.4.1. ログの収集

oc adm must-gather CLI を使用して、デバッグおよびトラブルシューティング用の情報を収集できます。

手順

- 次のコマンドを実行して、Operator に関するデータを収集します。

```
$  oc adm must-gather \
--dest-dir=must-gather/tmp \
--image=$(oc -n openshift-lifecycle-agent get deployment.apps/lifecycle-agent-
controller-manager -o jsonpath='{.spec.template.spec.containers[?(@.name ==
```

```
"manager")].image}') \
--image=quay.io/konveyor/oadp-must-gather:latest \ 1
--image=quay.io/openshift/origin-must-gather:latest 2
```

1 （オプション）OADP Operator からさらに情報を収集する必要がある場合は、このオプションを追加できます。

2 （オプション）SR-IOV Operator からより多くの情報を収集する必要がある場合は、このオプションを追加できます。

15.3.4.2. AbortFailed または FinalizeFailed エラー

問題

最終段階中、または Prep 段階でプロセスを停止すると、ライフサイクル環境は次のリソースをクリーンアップします。

- 不要になった Stateroot
- リソースの事前キャッシュ
- OADP CR
- ImageBasedUpgrade CR

Lifecycle Agent が上記の手順の実行に失敗すると、AbortFailed または FinalizeFailed 状態に移行します。状態メッセージとログは、失敗したステップを示しています。

エラーメッセージの例

```
message: failed to delete all the backup CRs. Perform cleanup manually then add
'lca.openshift.io/manual-cleanup-done' annotation to ibu CR to transition back to Idle
    observedGeneration: 5
```

```
reason: AbortFailed
status: "False"
type: Idle
```

解決方法

1. ログを調べて、障害が発生した理由を確認します。

2. Lifecycle Agent にクリーンアップを再試行するように要求するには、lca.openshift.io/manual-cleanup-done アノテーションを ImageBasedUpgrade CR に追加します。

   このアノテーションを確認した後、ライフサイクルエージェントはクリーンアップを再試行し、成功した場合、ImageBasedUpgrade ステージは Idle に移行します。

   クリーンアップが再び失敗する場合は、リソースを手動でクリーンアップできます。

15.3.4.2.1. stateroot の手動クリーンアップ

問題

Prep 段階で停止すると、Lifecycle Agent は新しい stateroot をクリーンアップします。アップグレードの正常な完了またはロールバック後にファイナライズすると、Lifecycle Agent は古い状態ルートをクリーンアップします。この手順が失敗した場合は、ログを調べて、障害が発生した理由を特定することが推奨されます。

解決方法

1. 次のコマンドを実行して、stateroot に既存のデプロイメントがあるかどうかを確認します。

   ```
   $ ostree admin status
   ```

2. 存在する場合は、次のコマンドを実行して既存のデプロイメントをクリーンアップします。

   ```
   $ ostree admin undeploy <index_of_deployment>
   ```

3.

stateroot のすべてのデプロイメントをクリーンアップした後に、次のコマンドを実行して stateroot ディレクトリーを消去します。

警告

起動したデプロイメントがこの stateroot にないことを確認します。

```
$ stateroot="<stateroot_to_delete>"
```

```
$ unshare -m /bin/sh -c "mount -o remount,rw /sysroot && rm -rf
/sysroot/ostree/deploy/${stateroot}"
```

15.3.4.2.2. OADP リソースの手動によるクリーンアップ

問題

OADP リソースの自動クリーンアップは、ライフサイクルエージェントと S3 バックエンド間の接続問題が原因で失敗する可能性があります。接続を復元し、lca.openshift.io/manual-cleanup-done アノテーションを追加すると、ライフサイクルエージェントはバックアップリソースを正常にクリーンアップできます。

解決方法

1. 次のコマンドを実行して、バックエンドの接続性を確認します。

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

出力例

```
NAME                          PHASE       LAST VALIDATED   AGE   DEFAULT
dataprotectionapplication-1   Available   33s              8d    true
```

2. すべてのバックアップリソースを削除してから、lca.openshift.io/manual-cleanup-done

アノテーションを ImageBasedUpgrade CR に追加します。

15.3.4.3. LVM ストレージボリュームのコンテンツが復元されない

LVM ストレージを使用して動的永続ボリュームストレージを提供する場合は、永続ボリュームの内容が正しく設定されていない場合、LVM ストレージが永続ボリュームのコンテンツを復元しない場合があります。

15.3.4.3.1. Backup CR に LVM ストレージ関連フィールドがない

問題

Backup CR には、永続ボリュームを復元するために必要なフィールドが欠落している可能性があります。アプリケーション Pod のイベントを確認し、以下を実行してこの問題の有無を判別できます。

```
$ oc describe pod <your_app_name>
```

Backup CR に LVM ストレージ関連のフィールドが欠落していることを示す出力例

```
Events:
  Type     Reason           Age              From               Message
  ----     ------           ----             ----               -------
  Warning  FailedScheduling  58s (x2 over 66s)  default-scheduler  0/1 nodes are available:
pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available:
1 Preemption is not helpful for scheduling..
  Normal   Scheduled         56s                default-scheduler  Successfully assigned default/db-
1234 to sno1.example.lab
  Warning  FailedMount       24s (x7 over 55s)  kubelet            MountVolume.SetUp failed for
volume "pvc-1234" : rpc error: code = Unknown desc = VolumeID is not found
```

解決方法

アプリケーションの Backup CR には logicalvolumes.topolvm.io を含める必要があります。このリソースがないと、アプリケーションは永続ボリューム要求と永続ボリュームマニフェストを正常に復元しますが、この永続 ボリュームに関連付けられた論理 ボリュームはピボット後に適切に復元されません。

Backup CR の例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
    velero.io/storage-location: default
  name: small-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  includedClusterScopedResources: 1
  - persistentVolumes
  - volumesnapshotcontents
  - logicalvolumes.topolvm.io
```

1

アプリケーションの永続ボリュームを復元するには、本セクションを以下のように設定する必要があります。

15.3.4.3.2. Restore CR に LVM ストレージ関連フィールドがない

問題

アプリケーションの予想されるリソースは復元されますが、永続ボリュームの内容はアップグレード後に保持されません。

1. ピボットの前に以下のコマンドを実行して、アプリケーションの永続ボリュームを一覧表示します。

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

ピボット前の出力例

```
NAME                     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Retain           Bound    default/pvc-
```

```
db   lvms-vg1            4h45m

NAMESPACE   NAME                  STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi      RWO
lvms-vg1      4h45m

NAMESPACE   NAME                         AGE
        logicalvolume.topolvm.io/pvc-1234   4h45m
```

2. ピボット後に次のコマンドを実行して、アプリケーションの永続ボリュームを一覧表示します。

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

ピボット後の出力例

```
NAME                 CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi      RWO          Delete         Bound    default/pvc-
db   lvms-vg1             19s

NAMESPACE   NAME                  STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi      RWO
lvms-vg1      19s

NAMESPACE   NAME                         AGE
        logicalvolume.topolvm.io/pvc-1234   18s
```

解決方法

この問題の理由は、論理ボリューム ステータスが Restore CR に保持されないためです。このステータスは、Velero がピボット後に保持する必要があるボリュームを参照する必要があるため、重要です。アプリケーションの Restore CR に以下のフィールドを含める必要があります。

Restore CR の例

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: sample-vote-app
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3"
spec:
  backupName:
    sample-vote-app
  restorePVs: true (1)
  restoreStatus: (2)
    includedResources:
      - logicalvolumes
```

(1) アプリケーションの永続ボリュームを保持するには、restorePVs を true に設定する必要があります。

(2) アプリケーションの永続ボリュームを保持するには、本セクションを以下のように設定する必要があります。

15.3.4.4. デバッグに失敗した Backup CR および Restore CR

問題

アーティファクトのバックアップまたは復元に失敗しました。

解決方法

Velero CLI ツールを使用して、Backup CR および Restore CR をデバッグし、ログを取得できます。Velero CLI ツールは、OpenShift CLI ツールよりも詳細な情報を提供します。

1. 次のコマンドを実行して、エラーを含む Backup CR を記述します。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe
backup -n openshift-adp backup-acm-klusterlet --details
```

2. 次のコマンドを実行して、エラーを含む Restore CR を記述します。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe
restore -n openshift-adp restore-acm-klusterlet --details
```

3. 次のコマンドを実行して、バックアップされたリソースをローカルディレクトリーにダウンロードします。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero backup
download -n openshift-adp backup-acm-klusterlet -o ~/backup-acm-klusterlet.tar.gz
```

15.4. GITOPS ZTP を使用した単一ノード OPENSHIFT クラスターのイメージベースのアップグレードの実行

GitOps Zero Touch Provisioning (ZTP)を使用して、イメージベースのアップグレードで管理対象の単一ノード OpenShift クラスターをアップグレードできます。

Lifecycle Agent をクラスターにデプロイすると、ImageBasedUpgrade CR が自動的に作成されます。この CR を更新して、シードイメージのイメージリポジトリーを指定し、別のステージを通過します。

15.4.1. Lifecycle Agent と GitOps ZTP を使用したイメージベースのアップグレードの Prep 段階への移行

Lifecycle Agent をクラスターにデプロイすると、ImageBasedUpgrade CR が自動的に作成されます。この CR を更新して、シードイメージのイメージリポジトリーを指定し、別のステージを通過します。

前提条件

- イメージベースのアップグレードで使用されるリソースのポリシーおよび ConfigMap オブジェクトを作成します。詳細は、「GitOps ZTP を使用したイメージベースのアップグレード用の ConfigMap オブジェクトの作成」を参照してください。

手順

1. Prep、Upgrade、および Idle ステージのポリシーを、ibu-upgrade-ranGen.yaml という既存のグループ PolicyGenTemplate に追加します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: example-group-ibu
  namespace: "ztp-group"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  evaluationInterval: 1
    compliant: 10s
    noncompliant: 10s
  sourceFiles:
    - fileName: ConfigMapGeneric.yaml
      complianceType: mustonlyhave
      policyName: "oadp-cm-policy"
      metadata:
        name: oadp-cm
        namespace: openshift-adp
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "prep-stage-policy"
      spec:
        stage: Prep
        seedImageRef: 2
          version: "4.15.0"
          image: "quay.io/user/lca-seed:4.15.0"
          pullSecretRef:
            name: "<seed_pull_secret>"
        oadpContent: 3
        - name: "oadp-cm"
          namespace: "openshift-adp"
      status:
        conditions:
          - reason: Completed
            status: "True"
            type: PrepCompleted
            message: "Prep stage completed successfully"
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "upgrade-stage-policy"
      spec:
        stage: Upgrade
      status:
        conditions:
          - reason: Completed
            status: "True"
            type: UpgradeCompleted
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "finalize-stage-policy"
      complianceType: mustonlyhave
      spec:
        stage: Idle
    - fileName: ibu/ImageBasedUpgrade.yaml
      policyName: "finalize-stage-policy"
      status:
        conditions:
```

```
    - reason: Idle
      status: "True"
      type: Idle
```

1

準拠ポリシーおよび非準拠ポリシーのポリシー評価間隔。ポリシーのステータスが現在のアップグレードステータスを正確に反映するように、10s に設定します。

2

Prep の段階で、アップグレードのシードイメージ、OpenShift Container Platform バージョン、およびプルシークレットを定義します。

3

バックアップと復元に必要な OADP ConfigMap リソースを定義します。

2. 以下のコマンドを実行して、イメージベースのアップグレードに必要なポリシーが作成されていることを確認します。

```
$ oc get policies -n spoke1 | grep -E "example-group-ibu"
```

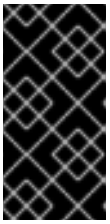
出力例

```
ztp-group.example-group-ibu-oadp-cm-policy       inform            NonCompliant
31h
ztp-group.example-group-ibu-prep-stage-policy        inform            NonCompliant
31h
ztp-group.example-group-ibu-upgrade-stage-policy     inform            NonCompliant
31h
ztp-group.example-group-ibu-finalize-stage-policy    inform            NonCompliant
31h
ztp-group.example-group-ibu-rollback-stage-policy    inform            NonCompliant
31h
```

3. du-profile クラスターラベルを、ターゲットプラットフォームバージョンまたは SiteConfig CR の対応する policy-binding ラベルに更新します。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
```

```
[...]
spec:
  [...]
    clusterLabels:
      du-profile: "4.15.0"
```



重要

ラベルをターゲットプラットフォームバージョンに更新すると、既存のポリシーセットがバインドが解除されます。

4. 更新された SiteConfig CR を Git リポジトリーにコミットしてプッシュします。

5. Prep 段階に移行する準備ができたら、Prep および OADP ConfigMap ポリシーを使用して、ターゲットハブクラスターに ClusterGroupUpgrade CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-oadp-cm-policy
  - example-group-ibu-prep-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

6. 以下のコマンドを実行して Prep ポリシーを適用します。

```
$ oc apply -f cgu-ibu-prep.yml
```

OADP リソースに ConfigMap オブジェクトと追加のマニフェストを指定した場合、ライフサイクルエージェントは、Prep の段階で指定された ConfigMap オブジェクトを検証します。以下の問題が発生する可能性があります。

- Lifecycle エージェントが extraManifestsの問題を検出した場合の検証の警告または

エラー

- oadpContentの問題をライフサイクルエージェントが検出した場合の検証エラー

検証の警告は Upgrade ステージをブロックしませんが、アップグレードを続行しても安全かどうかを判断する必要があります。CRD、namespace、ドライランの失敗の欠落など、これらの警告は、ImageBasedUpgrade CR の Prep stage および annotation フィールドの status.conditions を警告の詳細で更新します。

検証警告の例

```
[...]
metadata:
annotations:
  extra-manifest.lca.openshift.io/validation-warning: '...'
[...]
```

ただし、MachineConfig または Operator マニフェストを追加マニフェストに追加するなど、検証エラーにより、Prep 段階が失敗し、アップグレード ステージがブロックされます。

検証にパスすると、クラスターは新規の ostree stateroot を作成します。これには、シードイメージをプルおよびデプロイメントし、ホストレベルのコマンドを実行します。最後に、必要なすべてのイメージがターゲットクラスターに事前キャッシュされます。

7. 以下のコマンドを実行してステータスを監視し、cgu-ibu-prep ClusterGroupUpgrade が Completed を報告するのを待ちます。

```
$ oc get cgu -n default
```

出力例

```
NAME              AGE   STATE       DETAILS
cgu-ibu-prep        31h   Completed   All clusters are compliant with all the managed
policies
```

関連情報

- バージョンに依存しないように GitOps ZTP サイト設定リポジトリーを準備する
- GitOps ZTP を使用したライフサイクルエージェントによるイメージベースのアップグレード用の ConfigMap オブジェクトの作成
- GitOps ZTP を使用する場合の ostree stateroots 間で共有コンテナーディレクトリーの設定
- バックアップおよびスナップショットの場所、ならびにそのシークレットについて
- バックアップ CR の作成
- 復元 CR の作成

15.4.2. Lifecycle Agent と GitOps ZTP を使用したイメージベースのアップグレードのアップグレードステージへの移行

Prep 段階が完了したら、ターゲットクラスターをアップグレードできます。アップグレードプロセス中に、OADP Operator は OADP CR で指定されたアーティファクトのバックアップを作成し、ライフサイクルエージェントがクラスターをアップグレードします。

アップグレードが失敗するか停止すると、自動ロールバックが開始されます。アップグレード後に問題が発生した場合は、手動のロールバックを開始できます。手動ロールバックの詳細については、（オプション）ライフサイクルエージェントと GitOps ZTP を使用したロールバックの開始 を参照してください。

前提条件

- Prep ステージを完了します。

手順

1. Upgrade の段階に移行する準備ができたら、アップグレードポリシーを参照するターゲットハブクラスターに ClusterGroupUpgrade CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-upgrade
  namespace: default
spec:
  actions:
    beforeEnable:
      addClusterAnnotations:
        import.open-cluster-management.io/disable-auto-import: "true" (1)
    afterCompletion:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import (2)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-upgrade-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

(1) アップグレードを開始する前に、disable-auto-import アノテーションをマネージドクラスターに適用します。このアノテーションを使用すると、クラスターが準備状態になるまで、アップグレード段階でマネージドクラスターの自動インポートが無効になります。

(2) アップグレードの完了後に disable-auto-import アノテーションを削除します。

2. 以下のコマンドを実行して Upgrade ポリシーを適用します。

```
$ oc apply -f cgu-ibu-upgrade.yml
```

3. 以下のコマンドを実行してステータスを監視し、cgu-ibu-upgrade ClusterGroupUpgrade が Completed を報告するまで待機します。

```
$ oc get cgu -n default
```

出力例

```
NAME                    AGE   STATE      DETAILS
cgu-ibu-prep            31h   Completed  All clusters are compliant with all the managed
policies
cgu-ibu-upgrade         31h   Completed  All clusters are compliant with all the managed
policies
```

4. 変更に問題があり、アップグレードをファイナライズしたら、アップグレードを完了するポリシーを参照するターゲットハブクラスターに ClusterGroupUpgrade CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-finalize
  namespace: default
spec:
  actions:
    beforeEnable:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-finalize-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

重要

TALM が継続的に調整するため、他の ClusterGroupUpgrade CR が進行中ではないことを確認してください。cgu-ibu -finalize.yaml を適用する前に、すべての In-Progress の ClusterGroupUpgrade CR を削除します。

5. 以下のコマンドを実行してポリシーを適用します。

```
$ oc apply -f cgu-ibu-finalize.yaml
```

6. 以下のコマンドを実行してステータスをモニターし、cgu-ibu-finalize ClusterGroupUpgrade が Completed を報告するまで待機します。

```
$ oc get cgu -n default
```

出力例

```
NAME                AGE   STATE       DETAILS
cgu-ibu-finalize      30h   Completed   All clusters are compliant with all the managed policies
cgu-ibu-prep          31h   Completed   All clusters are compliant with all the managed policies
cgu-ibu-upgrade       31h   Completed   All clusters are compliant with all the managed policies
```

関連情報

- （オプション）ライフサイクルエージェントと GitOps ZTP を使用したイメージベースのアップグレードのロールバックステージへの移行

15.4.3. （オプション）Lifecycle Agent および GitOps ZTP を使用した Rollback ステージへの移行

アップグレード後に問題が発生した場合は、手動のロールバックを開始できます。

手順

1. du-profile または対応する policy-binding ラベルを、SiteConfig CR の元のプラットフォームバージョンに戻します。

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
[...]
spec:
  [...]
    clusterLabels:
      du-profile: "4.14.x"
```

2. ロールバックを開始する準備ができたら、Rollback ポリシーを既存のグループ

PolicyGenTemplate CR に追加します。

```
[...]
- fileName: ibu/ImageBasedUpgrade.yaml
  policyName: "rollback-stage-policy"
  spec:
    stage: Rollback
  status:
    conditions:
      - message: Rollback completed
        reason: Completed
        status: "True"
        type: RollbackCompleted
```

3. Rollback ポリシーを参照するターゲットハブクラスターに ClusterGroupUpgrade CR を作成します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-ibu-rollback
  namespace: default
spec:
  actions:
    beforeEnable:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-rollback-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

4. 以下のコマンドを実行して Rollback ポリシーを適用します。

```
$ oc apply -f cgu-ibu-rollback.yml
```

5. 変更内容に問題がなければ、ロールバックを完了する準備ができたら、ターゲットハブクラスターに ClusterGroupUpgrade CR を作成します。この CR は、ロールバックをファイナライズするポリシーを参照します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
```

```
metadata:
  name: cgu-ibu-finalize
  namespace: default
spec:
  actions:
    beforeEnable:
      removeClusterAnnotations:
      - import.open-cluster-management.io/disable-auto-import
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - example-group-ibu-finalize-stage-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 1
    timeout: 240
```

6. 以下のコマンドを実行してポリシーを適用します。

```
$ oc apply -f cgu-ibu-finalize.yml
```

15.4.4. Lifecycle Agent を使用したイメージベースのアップグレードのトラブルシューティング

イメージベースのアップグレード中に問題が発生する可能性があります。

15.4.4.1. ログの収集

oc adm must-gather CLI を使用して、デバッグおよびトラブルシューティング用の情報を収集できます。

手順

- 次のコマンドを実行して、Operator に関するデータを収集します。

```
$  oc adm must-gather \
--dest-dir=must-gather/tmp \
--image=$(oc -n openshift-lifecycle-agent get deployment.apps/lifecycle-agent-controller-manager -o jsonpath='{.spec.template.spec.containers[?(@.name == "manager")].image}') \
--image=quay.io/konveyor/oadp-must-gather:latest \ 1
--image=quay.io/openshift/origin-must-gather:latest 2
```

1

2

（オプション）SR-IOV Operator からより多くの情報を収集する必要がある場合は、このオプションを追加できます。

15.4.4.2. AbortFailed または FinalizeFailed エラー

問題

最終段階中、または Prep 段階でプロセスを停止すると、ライフサイクル環境は次のリソースをクリーンアップします。

- 不要になった Stateroot
- リソースの事前キャッシュ
- OADP CR
- ImageBasedUpgrade CR

Lifecycle Agent が上記の手順の実行に失敗すると、AbortFailed または FinalizeFailed 状態に移行します。状態メッセージとログは、失敗したステップを示しています。

エラーメッセージの例

```
message: failed to delete all the backup CRs. Perform cleanup manually then add
'lca.openshift.io/manual-cleanup-done' annotation to ibu CR to transition back to Idle
    observedGeneration: 5
    reason: AbortFailed
    status: "False"
    type: Idle
```

解決方法

1. ログを調べて、障害が発生した理由を確認します。

2. Lifecycle Agent にクリーンアップを再試行するように要求するには、lca.openshift.io/manual-cleanup-done アノテーションを ImageBasedUpgrade CR に追加します。

このアノテーションを確認した後、ライフサイクルエージェントはクリーンアップを再試行し、成功した場合、ImageBasedUpgrade ステージは Idle に移行します。

クリーンアップが再び失敗する場合は、リソースを手動でクリーンアップできます。

15.4.4.2.1. stateroot の手動クリーンアップ

問題

Prep 段階で停止すると、Lifecycle Agent は新しい stateroot をクリーンアップします。アップグレードの正常な完了またはロールバック後にファイナライズすると、Lifecycle Agent は古い状態ルートをクリーンアップします。この手順が失敗した場合は、ログを調べて、障害が発生した理由を特定することが推奨されます。

解決方法

1. 次のコマンドを実行して、stateroot に既存のデプロイメントがあるかどうかを確認します。

```
$ ostree admin status
```

2. 存在する場合は、次のコマンドを実行して既存のデプロイメントをクリーンアップします。

```
$ ostree admin undeploy <index_of_deployment>
```

3. stateroot のすべてのデプロイメントをクリーンアップした後に、次のコマンドを実行して stateroot ディレクトリーを消去します。

警告

起動したデプロイメントがこの stateroot にないことを確認します。

```
$ stateroot="<stateroot_to_delete>"
```

```
$ unshare -m /bin/sh -c "mount -o remount,rw /sysroot && rm -rf
/sysroot/ostree/deploy/${stateroot}"
```

15.4.4.2.2. OADP リソースの手動によるクリーンアップ

問題

OADP リソースの自動クリーンアップは、ライフサイクルエージェントと S3 バックエンド間の接続問題が原因で失敗する可能性があります。接続を復元し、lca.openshift.io/manual-cleanup-done アノテーションを追加すると、ライフサイクルエージェントはバックアップリソースを正常にクリーンアップできます。

解決方法

1. 次のコマンドを実行して、バックエンドの接続性を確認します。

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

出力例

```
NAME                           PHASE       LAST VALIDATED   AGE   DEFAULT
dataprotectionapplication-1    Available   33s              8d    true
```

2. すべてのバックアップリソースを削除してから、lca.openshift.io/manual-cleanup-done アノテーションを ImageBasedUpgrade CR に追加します。

15.4.4.3. LVM ストレージボリュームのコンテンツが復元されない

LVM ストレージを使用して動的永続ボリュームストレージを提供する場合は、永続ボリュームの内容が正しく設定されていない場合、LVM ストレージが永続ボリュームのコンテンツを復元しない場合があります。

15.4.4.3.1. Backup CR に LVM ストレージ関連フィールドがない

問題

Backup CR には、永続ボリュームを復元するために必要なフィールドが欠落している可能性があります。アプリケーション Pod のイベントを確認し、以下を実行してこの問題の有無を判別できます。

```
$ oc describe pod <your_app_name>
```

Backup CR に LVM ストレージ関連のフィールドが欠落していることを示す出力例

```
Events:
  Type     Reason          Age              From               Message
  ----     ------          ----             ----               -------
  Warning  FailedScheduling  58s (x2 over 66s)  default-scheduler  0/1 nodes are available:
pod has unbound immediate PersistentVolumeClaims. preemption: 0/1 nodes are available:
1 Preemption is not helpful for scheduling..
  Normal   Scheduled         56s                default-scheduler  Successfully assigned default/db-
1234 to sno1.example.lab
  Warning  FailedMount       24s (x7 over 55s)  kubelet            MountVolume.SetUp failed for
volume "pvc-1234" : rpc error: code = Unknown desc = VolumeID is not found
```

解決方法

アプリケーションの Backup CR には logicalvolumes.topolvm.io を含める必要があります。このリソースがないと、アプリケーションは永続ボリューム要求と永続ボリュームマニフェストを正常に復元しますが、この永続 ボリュームに関連付けられた論理 ボリュームはピボット後に適切に復元されません。

Backup CR の例

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  labels:
```

```
    velero.io/storage-location: default
  name: small-app
  namespace: openshift-adp
spec:
  includedNamespaces:
  - test
  includedNamespaceScopedResources:
  - secrets
  - persistentvolumeclaims
  - deployments
  - statefulsets
  includedClusterScopedResources: 1
  - persistentVolumes
  - volumesnapshotcontents
  - logicalvolumes.topolvm.io
```

1

アプリケーションの永続ボリュームを復元するには、本セクションを以下のように設定する必要があります。

15.4.4.3.2. Restore CR に LVM ストレージ関連フィールドがない

問題

アプリケーションの予想されるリソースは復元されますが、永続ボリュームの内容はアップグレード後に保持されません。

1. ピボットの前に以下のコマンドを実行して、アプリケーションの永続ボリュームを一覧表示します。

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

ピボット前の出力例

```
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Retain           Bound    default/pvc-
db   lvms-vg1              4h45m

NAMESPACE   NAME                  STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
```

```
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       4h45m

NAMESPACE   NAME                            AGE
            logicalvolume.topolvm.io/pvc-1234   4h45m
```

2. ピボット後に次のコマンドを実行して、アプリケーションの永続ボリュームを一覧表示します。

```
$ oc get pv,pvc,logicalvolumes.topolvm.io -A
```

ピボット後の出力例

```
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM          STORAGECLASS   REASON   AGE
persistentvolume/pvc-1234   1Gi        RWO            Delete           Bound    default/pvc-
db   lvms-vg1              19s

NAMESPACE   NAME                  STATUS   VOLUME    CAPACITY   ACCESS
MODES   STORAGECLASS   AGE
default    persistentvolumeclaim/pvc-db   Bound    pvc-1234   1Gi        RWO
lvms-vg1       19s

NAMESPACE   NAME                            AGE
            logicalvolume.topolvm.io/pvc-1234   18s
```

解決方法

この問題の理由は、論理ボリューム ステータスが Restore CR に保持されないためです。このステータスは、Velero がピボット後に保持する必要があるボリュームを参照する必要があるため、重要です。アプリケーションの Restore CR に以下のフィールドを含める必要があります。

Restore CR の例

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: sample-vote-app
```

```
  namespace: openshift-adp
  labels:
    velero.io/storage-location: default
  annotations:
    lca.openshift.io/apply-wave: "3"
spec:
  backupName:
    sample-vote-app
  restorePVs: true (1)
  restoreStatus: (2)
    includedResources:
      - logicalvolumes
```

(1) アプリケーションの永続ボリュームを保持するには、restorePVs を true に設定する必要があります。

(2) アプリケーションの永続ボリュームを保持するには、本セクションを以下のように設定する必要があります。

15.4.4.4. デバッグに失敗した Backup CR および Restore CR

問題

アーティファクトのバックアップまたは復元に失敗しました。

解決方法

Velero CLI ツールを使用して、Backup CR および Restore CR をデバッグし、ログを取得できます。Velero CLI ツールは、OpenShift CLI ツールよりも詳細な情報を提供します。

1. 次のコマンドを実行して、エラーを含む Backup CR を記述します。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe
backup -n openshift-adp backup-acm-klusterlet --details
```

2. 次のコマンドを実行して、エラーを含む Restore CR を記述します。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero describe
restore -n openshift-adp restore-acm-klusterlet --details
```

3. 次のコマンドを実行して、バックアップされたリソースをローカルディレクトリーにダウンロードします。

```
$ oc exec -n openshift-adp velero-7c87d58c7b-sw6fc -c velero -- ./velero backup
download -n openshift-adp backup-acm-klusterlet -o ~/backup-acm-klusterlet.tar.gz
```