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: 2025-12-06

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

Legal Notice

Copyright © 2025 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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、ネットワークのファーエッジのサイトをプロビジョニングおよび管理する方法を説明します。

Table of Contents

第1章 ネットワークファーエッジの課題 5
1.1. ネットワークファーエッジの課題の解決 5
1.2. GITOPS ZTP を使用したネットワークファーエッジでのクラスタープロビジョニング 6
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 7
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 8

第2章 GITOPS ZTP 用のハブクラスターの準備 11
2.1. TELCO RAN DU 4.16 の検証済みソフトウェアコンポーネント 11
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 12
2.3. 非接続環境での GITOPS ZTP のインストール 13
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 14
2.5. アシストサービスの有効化 15
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 16
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 19
2.8. ARGOCD を使用したハブクラスターの設定 20
2.9. GITOPS ZTP サイト設定リポジトリーの準備 23
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 25

第3章 GITOPS ZTP の更新 29
3.1. GITOPS ZTP 更新プロセスの概要 29
3.2. アップグレードの準備 30
3.3. 既存クラスターのラベル付け 30
3.4. 既存の GITOPS ZTP アプリケーションの停止 31
3.5. GIT リポジトリーに必要な変更 31
3.6. 新規 GITOPS ZTP アプリケーションのインストール 33
3.7. 必要な OPENSHIFT CONTAINER PLATFORM バージョンの ISO イメージのプル 35
3.8. GITOPS ZTP 設定の変更のロールアウト 36

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 37
4.1. GITOPS ZTP および TOPOLOGY AWARE LIFECYCLE MANAGER 37
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 39
4.3. マネージドベアメタルホストシークレットの作成 40
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 41
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 43
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 57
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 58
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 59
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 60
4.10. GITOPS ZTP パイプラインからの古いコンテンツの削除 60
4.11. GITOPS ZTP パイプラインの破棄 61

第5章 GITOPS ZTP を使用したシングルノード OPENSHIFT クラスターの手動インストール 62
5.1. GITOPS ZTP インストール CR と設定 CR の手動生成 62
5.2. マネージドベアメタルホストシークレットの作成 68
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 69
5.4. 単一のマネージドクラスターのインストール 71
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 72
5.6. マネージドクラスターのトラブルシューティング 73
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 74

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シングルノード OPENSHIFT クラスター設定 76
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 76

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 76
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 77
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 78
6.5. GITOPS ZTP を使用したシングルノード OPENSHIFT でのワークロードの分割 78
6.6. 推奨されるクラスターインストールマニフェスト 79
6.7. 推奨されるインストール後のクラスター設定 89

第7章 VDU アプリケーションワークロードのシングルノード OPENSHIFT クラスターチューニングの検証 114
7.1. VDU クラスターホストの推奨ファームウェア設定 114
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 116
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 120

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 130
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 130
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 131
8.3. SITECONFIG CR を使用してノードを削除する 133

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 136
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 136
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 154
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 183

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 209
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 209
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 223
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 250

第11章 POLICYGENERATOR または POLICYGENTEMPLATE CR でのハブテンプレートの使用 276
11.1. グループ POLICYGENERATOR または POLICYGENTEMPLATE CR でのグループとサイト設定の指定 276
11.2. 新しい CONFIGMAP の変更を既存の POLICYGENERATOR または POLICYGENTEMPLATE CR に同期する 283

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 285
12.1. TOPOLOGY AWARE LIFECYCLE MANAGER の設定について 285
12.2. TOPOLOGY AWARE LIFECYCLE MANAGER で使用される管理ポリシー 285
12.3. WEB コンソールを使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール 286
12.4. CLI を使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール 287
12.5. CLUSTERGROUPUPGRADE CR 288
12.6. マネージドクラスターでのポリシーの更新 303
12.7. コンテナーイメージ事前キャッシュ機能の使用 312
12.8. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング 317

第13章 GITOPS ZTP を使用したシングルノード OPENSHIFT クラスターの拡張 326
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 327
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 327
13.3. PTP および SR-IOV ノードセレクターの互換性 328
13.4. POLICYGENERATOR CR を使用したワーカーノードポリシーのワーカーノードへの適用 329
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 331
13.6. GITOPS ZTP を使用してシングルノード OPENSHIFT クラスターにワーカーノードを追加する 334

第14章 シングルノード OPENSHIFT デプロイメント用イメージの事前キャッシュ 338
14.1. FACTORY-PRECACHING-CLI ツールの入手 338
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 339
14.3. ディスクのパーティション設定 340

14.4. イメージのダウンロード 344
14.5. GITOPS ZTP でのイメージの事前キャッシュ 353
14.6. "RENDERED CATALOG IS INVALID" エラーのトラブルシューティング 359

第15章 シングルノード OPENSHIFT クラスターのイメージベースアップグレード 361
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 361
15.2. シングルノード OPENSHIFT クラスターのイメージベースアップグレードの準備 372
15.3. LIFECYCLE AGENT を使用したシングルノード OPENSHIFT クラスターのイメージベースアップグレードの実行 408
15.4. GITOPS ZTP を使用したシングルノード OPENSHIFT クラスターのイメージベースアップグレードの実行 420

第16章 通信事業者コア CNF クラスターの DAY 2 オペレーション 440
16.1. 通信事業者向けコア CNF クラスターのアップグレード 440
16.2. 通信事業者コア CNF クラスターのトラブルシューティングとメンテナンス 493
16.3. 可観測性 510
16.4. セキュリティー 522

第1章 ネットワークファーエッジの課題

エッジコンピューティングでは、地理的に離れた場所にある多数のサイトを管理する際に複雑な課題が生じます。ネットワークのファーエッジのサイトをプロビジョニングおよび管理するには、GitOps Zero Touch Provisioning (ZTP) を使用します。

1.1. ネットワークファーエッジの課題の解決

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?
- 非接続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?

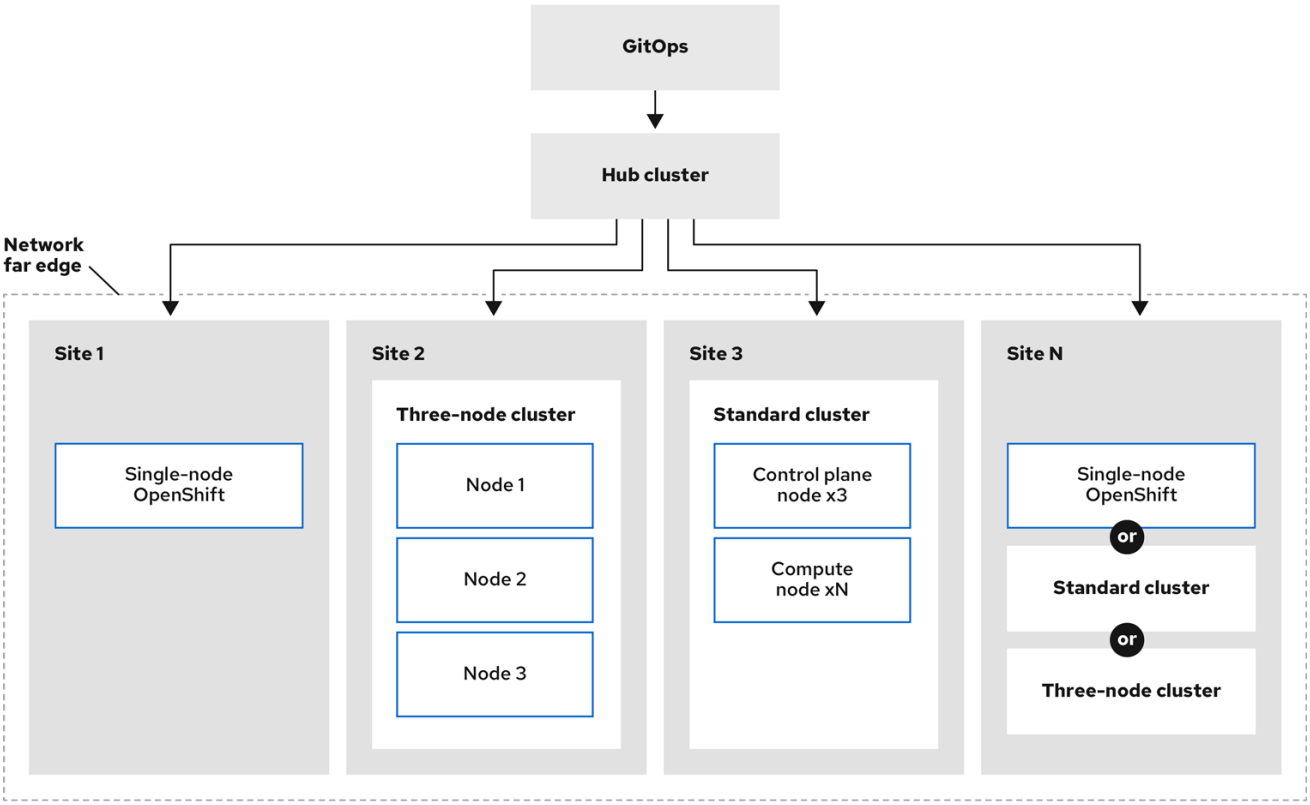
GitOps Zero Touch Provisioning (ZTP) と **GitOps** は、ベアメタル機器の宣言的なサイト定義と設定を使用してリモートエッジサイトを大規模にプロビジョニングできるようにすることで、これらの課題を解決します。テンプレートまたはオーバーレイ設定は、CNF ワークロードに必要な OpenShift Container Platform 機能をインストールします。インストールとアップグレードの全ライフサイクルは、GitOps ZTP パイプラインを通じて処理されます。

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、ファーエッジ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ファーエッジでのクラスタープロビジョニング

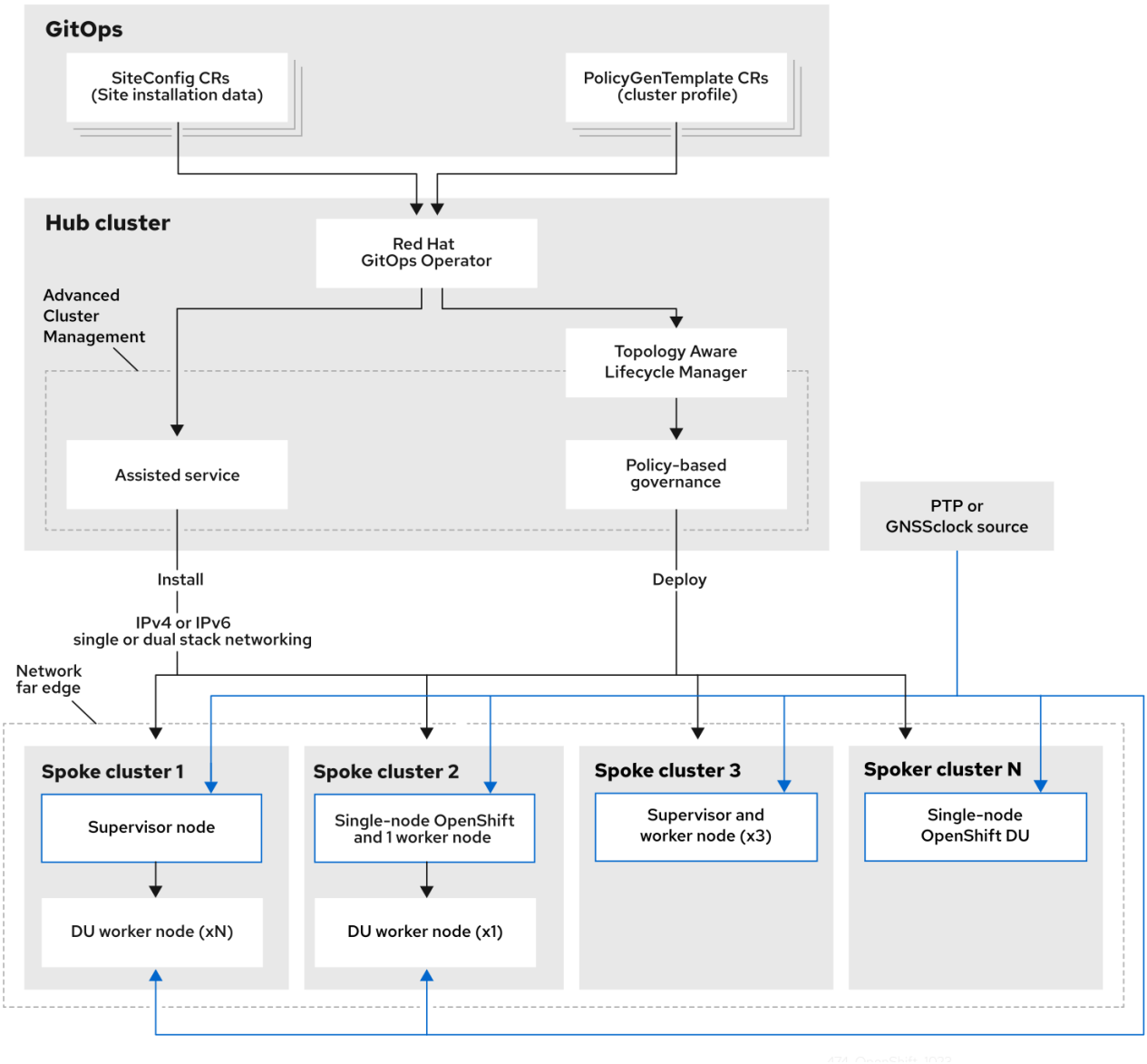
Red Hat Advanced Cluster Management (RHACM) は、単一のハブクラスターが多数のスポーククラスターを管理するハブアンドスポークアーキテクチャーでクラスターを管理します。RHACM を実行するハブクラスターは、GitOps Zero Touch Provisioning (ZTP) と、RHACM のインストール時にデプロイされるアシストサービスを使用して、マネージドクラスターをプロビジョニングおよびデプロイします。

アシストサービスは、ベアメタルで実行されるシングルノードクラスター、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は、最初のインストールフェーズと、その後の設定およびデプロイフェーズを含む 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。

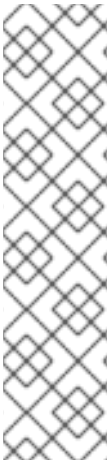
多くのクラスターのプロビジョニング

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。

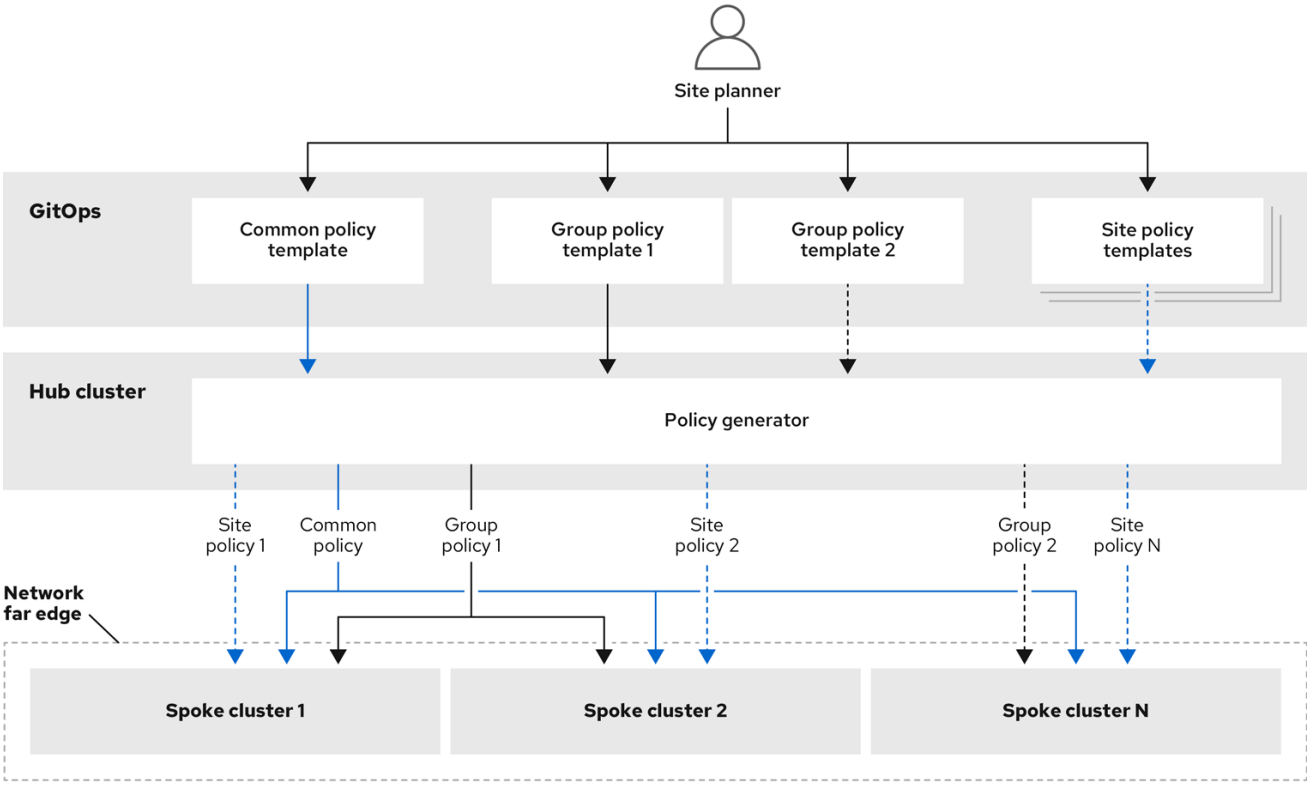


注記

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 **PolicyGenerator** 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 **PolicyGenerator** CR を使用して、シングルノード、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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較
- 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

非接続環境で 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 4.16 の検証済みソフトウェアコンポーネント

Red Hat telco RAN DU 4.16 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 通信事業者 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	6.0
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)	2.10, 2.11
Red Hat OpenShift GitOps	1.16
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。

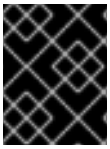
ワークロードのモニタリングと管理

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。

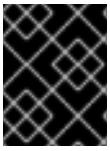


重要

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
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. 非接続環境での GITOPS ZTP のインストール

非接続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

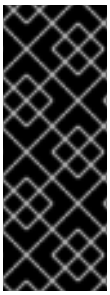
- OpenShift GitOps のインストール
- TALM のインストール
- Operator カタログのミラーリング

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-4.16.1-x86_64-live.x86_64.iso**)

1 RootFS イメージ名 (例: **rhcos-4.16.1-x86_64-live-rootfs.x86_64.img**)

1 OpenShift Container Platform バージョン (例: **4.16.1**)

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. アシストサービスの有効化

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、アシストサービスが自動的にデプロイされます。その後、すべての namespace 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. **spec.osImages** フィールドを更新するために、次のコマンドを実行して **AgentServiceConfig** CR を開きます。

```
$ oc edit AgentServiceConfig
```

3. **AgentServiceConfig** CR の **spec.osImages** フィールドを更新します。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
 name: agent
spec:
# ...
  osImages:
    - cpuArchitecture: x86_64
      openshiftVersion: "4.16"
      rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
      url: https://<host>/<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) を指定します。

<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を指定します。

4. 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

非接続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) 2.11 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**

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-**

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

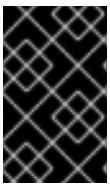
```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  namespace: multicluster-engine 1
spec:
  databaseStorage:
    volumeName: <db_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <db_storage_size>
  filesystemStorage:
    volumeName: <fs_pv_name>
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: <fs_storage_size>
  mirrorRegistryRef:
    name: assisted-installer-mirror-config 2
  osImages:
    - openshiftVersion: <ocp_version> 3
      url: <iso_url> 4
```

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 OpenShift Container Platform のバージョンを x.y または x.y.z 形式に設定します。

4 **httpd** サーバーでホストされている 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>

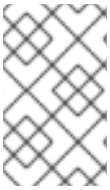
unauthenticated-image-registry.openshift-image-registry.svc:5000 などの新しいレジストリーです。

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** と認証情報を指定します。
      - **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。

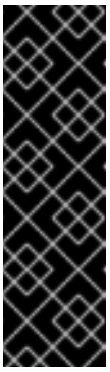
c. 2 つの ArgoCD アプリケーション、**out/argocd/deployment/clusters-app.yaml** と **out/argocd/deployment/policies-app.yaml** を、Git リポジトリーに基づいて修正します。

- 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** 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。

- RHACM 2.8 および 2.9 の場合は、**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v<rhacm_version>** イメージを使用します。
- RHACM 2.10 以降の場合は、**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v<rhacm_version>** イメージを使用します。



重要

multicluster-operators-subscription イメージのバージョンは、RHACM の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。

OpenShift Operator のライフサイクル の表「Platform Aligned Operators」の **[Expand for Operator list]** をクリックすると、OpenShift Container Platform でサポートされている Operator の完全なマトリックスが表示されます。

b. RHACM バージョンに一致する **multicluster-operators-subscription** イメージを使用して、**out/argocd/deployment/argocd-openshift-gitops-patch.json** ファイルを変更します。

```
{
  "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
```

5. オプション: 既存の ArgoCD アプリケーションがある場合は、次のコマンドを実行して、**Application** リソースに **PrunePropagationPolicy=background** ポリシーが設定されていることを確認します。

```
$ oc -n openshift-gitops get applications.argoproj.io  \
clusters -o jsonpath='{.spec.syncPolicy.syncOptions}' |jq
```

既存のポリシーの出力例

```
[
 "CreateNamespace=true",
 "PrunePropagationPolicy=background",
 "RespectIgnoreDifferences=true"
]
```

a. **spec.syncPolicy.syncOption** フィールドに **PrunePropagationPolicy** パラメーターが含まれていない場合、または **PrunePropagationPolicy** が **foreground** 値に設定されている

場合は、**Application** リソースでポリシーを **background** に設定します。以下の例を参照してください。

```
kind: Application
spec:
  syncPolicy:
    syncOptions:
    - PrunePropagationPolicy=background
```

background 削除ポリシーを設定すると、**ManagedCluster** CR とそれに関連付けられたすべてのリソースが削除されます。

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** と **PolicyGenerator** または **PolicyGentemplate** CR 用に個別のパスを持つディレクトリー構造を作成します。

注記

SiteConfig と **PolicyGenerator** または **PolicyGentemplate** CR を個別のディレクトリーに保存します。**SiteConfig** ディレクトリーと、**PolicyGenerator** または **PolicyGentemplate** ディレクトリーの両方に、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、**PolicyGenerator** 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、推奨される設定を表す **SiteConfig** ファイルと、**PolicyGenerator** または **PolicyGentemplate** ファイルのサンプルが含まれています。

4. **out/source-crs** フォルダーとその内容を **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 を使用してポリシーを管理およびデプロイし、クラスターを管理することは、OpenShift Container Platform の今後のリリースでは非推奨になります。Red Hat Advanced Cluster Management (RHACM) および **PolicyGenerator** CR を使用すると、同等の機能および改善された機能が利用できます。

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、シングルノード、3 ノード、および標準クラスターの **SiteConfig** と、**PolicyGenerator** または **PolicyGentemplate** 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、シングルノードクラスターのネットワークの CR のセットを説明しています。

```
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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

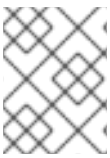
PolicyGenerator リソースの詳細は、RHACM Policy Generator のドキュメントを参照してください。

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 リソースのパッチ適用の比較

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

次の手順では、クラスターポリシー管理に **PolicyGentemplate** リソースではなく、**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 **PolicyGenerator** 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. **PolicyGenerator** 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 を使用した構造を説明します。

```
├── 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 **/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。

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。

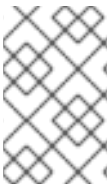
```
resources:
- version_4.13 (1)
#- version_4.14 (2)
```

(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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較

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ストラテジー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. 必要な OpenShift Container Platform バージョンの ISO イメージのプルを有効にします。
6. 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。

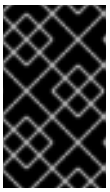
```
$ mkdir -p ./update
```

```
$ 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**: **PolicyGenerator** または **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** と、**PolicyGenerator** または **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

次の手順では、クラスターポリシー管理に **PolicyGentemplate** リソースではなく、**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** という接頭辞が付いた **Namespace** 内に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 **PolicyGenerator** 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表示されているファイルには、**SiteConfig** または **{policy-gen-cr}** CR のいずれか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** セクション内のすべての **PolicyGenerator** YAML ファイルと、**resources** セクションの **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

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、**pre-sync.yaml** ファイルと **post-sync.yaml** ファイルのセットが存在します。

- 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 **PolicyGenerator** 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。

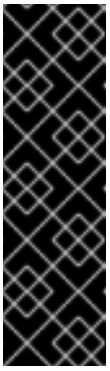
手順

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。

      - RHACM 2.8 および 2.9 の場合は、**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel8:v<rhacm_version>** イメージを使用します。

      - RHACM 2.10 以降の場合は、**registry.redhat.io/rhacm2/multicluster-operators-subscription-rhel9:v<rhacm_version>** イメージを使用します。



重要

multicluster-operators-subscription イメージのバージョンは、RHACM の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。

OpenShift Operator のライフサイクル の表「Platform Aligned Operators」の **[Expand for Operator list]** をクリックすると、OpenShift Container Platform でサポートされている Operator の完全なマトリックスが表示されます。

b. RHACM バージョンに一致する **multicluster-operators-subscription** イメージを使用して、**out/argocd/deployment/argocd-openshift-gitops-patch.json** ファイルを変更します。

```
{
  "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
```

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. 必要な OPENSHIFT CONTAINER PLATFORM バージョンの ISO イメージのプル

必要な OpenShift Container Platform バージョンの ISO イメージをプルするには、ミラーレジストリー HTTP サーバーでホストされている必要な 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 で **MultiClusterHub** が有効になっている。
- アシストサービスが有効になっている。

手順

1. **spec.osImages** フィールドを更新するために、次のコマンドを実行して **AgentServiceConfig** CR を開きます。

```
$ oc edit AgentServiceConfig
```

2. **AgentServiceConfig** CR の **spec.osImages** フィールドを更新します。

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
 name: agent
spec:
# ...
  osImages:
    - cpuArchitecture: x86_64
      openshiftVersion: "4.16"
      rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
      url: https://<host>/<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) を指定します。

<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を指定します。

3. エディターを保存して終了し、変更を適用します。

関連情報

- アシストサービスの有効化

3.8. 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) は、Topology Aware Lifecycle Manager の設定について を参照してください。
- **ClusterGroupUpgrade** CR の作成に関する詳細は、GitOps ZTP 用に自動作成された 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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較

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 は作成された各 **inform** ポリシーを確認し、これらのポリシーを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、修復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 GitOps ZTP インストールを再開できます。

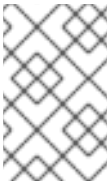
Waves

PolicyGenerator または **PolicyGentemplate** CR から生成された各ポリシーには、**ztp-deploy-wave** アノテーションが含まれます。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** でオーバーライドできます。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。



注記

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 設定が開始すると、**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、シングルノード 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。

4.2.1. 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。

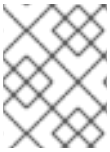
マネージドクラスターの **Agent** CR がハブクラスター上に作成されると、検出イメージ ISO プロセスが完了します。

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シングルノード 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 4.16 では、カーネル引数のみを追加できます。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。

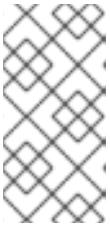
```
$ cat /proc/cmdline
```

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る。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。

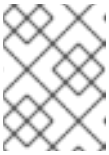
     a. 次のコマンドを実行して、クラスター namespace をエクスポートします。

     ```
     $ 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、シングルノード、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-4.16"
  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 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は、「関連情報」セクションを参照してください。この例では、読みやすくするために、**installConfigOverrides** フィールドと **ignitionConfigOverride** フィールドが展開されています。

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

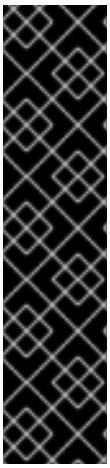
```
Name:   example-node.example.com
Roles:  control-plane,example-label,master,worker
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        custom-label/parameter1=true
        kubernetes.io/arch=amd64
        kubernetes.io/hostname=cnfdf03.telco5gran.eng.rdu2.redhat.com
        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 の高速プロビジョニングは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

シングルノード OpenShift 用の GitOps ZTP の高速プロビジョニングを使用すると、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 の高速プロビジョニングは、Assisted Installer を使用してシングルノード OpenShift をインストールする場合にのみサポートされます。これ以外の場合は、このインストール方法は失敗します。

4.5.1.1. 高速 ZTP のアクティブ化

以下の例のように、**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。

高速 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 を使用すると、高速化プロセスを完全に自動化できます。GitOps ZTP により、高速 GitOps ZTP **ConfigMap** への参照を使用して **AgentClusterInstall** リソースが更新され、TALM によってポリシーから抽出されたリソースと、高速 ZTP ジョブマニフェストが追加されます。

accelerated-ztp: partial を使用すると、GitOps ZTP により、高速ジョブマニフェストは追加されませんが、次の **kind** のクラスターインストール中に作成されたポリシーに基づくオブジェクトが追加されます。

- **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、**Performance Profile**、**Tuned**、および **ContainerRuntimeConfig** などの kind のリソースを適用するときに、ノードによって実行される再起動の回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、ポリシーに基づく Operator サブスクリプションをインストールします。

高速 ZTP の利点は、デプロイメントの規模に応じて増大します。**accelerated-ztp: full** を使用すると、多数のクラスターでより大きなメリットをもたらします。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 ZTP では、スポーク上に namespace と完了したジョブが残るため、手動で削除する必要があります。

accelerated-ztp: partial を使用する利点の 1 つは、ストック実装で問題が発生した場合や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オーバーライドできることです。

4.5.1.2. 高速 ZTP のプロセス

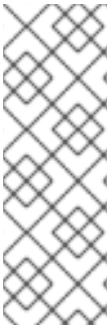
高速 ZTP は追加の **ConfigMap** を使用して、スポーククラスターのポリシーに基づくリソースを作成します。標準の **ConfigMap** には、GitOps ZTP ワークフローがクラスターのインストールをカスタマイズするために使用するマニフェストが含まれています。

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

TALM は 2 番目の **ConfigMap** を作成した後、マネージドクラスターにバインドされているすべてのポリシーを検出し、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、その 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シングルノード OpenShift クラスターの IPsec 暗号化の設定

GitOps ZTP と Red Hat Advanced Cluster Management (RHACM) を使用してインストールするシングルノード OpenShift マネージドクラスターで 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 と RHACM を使用した IPsec 暗号化のデプロイは、シングルノード OpenShift クラスターに対してのみ検証されます。

GitOps ZTP IPsec 実装では、リソースが制限された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、シングル **MachineConfig** CR のみを使用してこの機能をインストールし、前提条件としてシングルノード 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。
- **butane** ユーティリティー (バージョン 0.20.0 以降) がインストールされている。
- IPsec エンドポイント用の 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とマージします。
2. クラスター内の IPsec を設定するために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. **<cluster_node>** を、クラスター側 IPsec トンネルのクラスターノードの IP アドレスまたは DNS ホスト名に置き換えます。
2. **<left_cert>** を IPsec 証明書のニックネームに置き換えます。

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

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

5 IKEv2 VPN 暗号化プロトコルのみを使用します。IKEv1 は使用しないでください。これは非推奨となっています。

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、各ホストのネットワークセキュリティーサービス (NSS) データベースで必要です。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. カスタムサイト設定データを保持する 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ます。

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

   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 ネットワークに関連する **ca.pem** および **left_server.p12** 証明書ファイルを提供します。

6. カスタムサイト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、**extraManifests.searchPaths** フィールドに **custom-manifest/** ディレクトリーを追加します。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。Argo CD 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メントを開始します。

   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

シングルノード OpenShift のマネージドクラスターに IPsec 暗号化が正常に適用されていることを確認するには、次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. クラスターノードに IPsec ポリシーが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 トンネルが稼働し、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内の既知の IP に ping を実行します。たとえば、**ipsec/ipsec-endpoint-config.yaml** で設定した **rightsubnet** 範囲内の IP に ping を実行します。

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

出力例

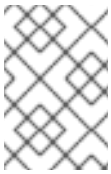
```
sh-5.1# ping 172.16.110.8
PING 172.16.110.8 (172.16.110.8) 56(84) bytes of data.
64 bytes from 172.16.110.8: icmp_seq=1 ttl=64 time=153 ms
64 bytes from 172.16.110.8: icmp_seq=2 ttl=64 time=155 ms
```

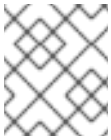
関連情報

- 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** が設定されている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: ""** が設定されている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	シングルノードの導入では、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。

SiteConfig CR フィールド	説明
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。たとえば、**wwn: <disk_wwn>** または **deviceName: /dev/disk/by-path/<device_path>** です。**<by-path>** 値が推奨されます。安定した識別子の詳細なリストは、「ルートデバイスヒントについて」セクションを参照してください。
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 アドレスを持つシングルノード 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、**PolicyGenerator** または **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 get applications.argoproj.io -n openshift-gitops clusters -o yaml
   ```

   a. マネージドクラスターのエラーログを識別するに

は、**status.operationState.syncResult.resources** フィールドを調べます。たとえば、**SiteConfig** CR の **extraManifestPath** に無効な値が割り当てられると、次のようなエラーが生成されます。

```
syncResult:
  resources:
  - group: ran.openshift.io
    kind: SiteConfig
    message: The Kubernetes API could not find ran.openshift.io/SiteConfig for
      requested resource spoke-sno/spoke-sno. Make sure the "SiteConfig" CRD is
      installed on the destination cluster
```

b. より詳細な **SiteConfig** エラーを表示するには、次の手順を実行します。

i. Argo CD ダッシュボードで、Argo CD が同期しようとしている **SiteConfig** リソースをクリックします。

ii. **DESIRED MANIFEST** タブをチェックして、**siteConfigError** フィールドを見つけます。

```
siteConfigError: >- Error: could not build the entire SiteConfig defined by /tmp/kust-
plugin-config-1081291903: stat sno-extra-manifest: no such file or directory
```

c. **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

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. **kustomization.yaml** ファイルから、関連する **SiteConfig** と、**PolicyGenerator** または **PolicyGentemplate** ファイルを削除して、サイトと関連する CR を削除します。
2. 次の **syncOptions** フィールドを **SiteConfig** アプリケーションに追加します。

```
kind: Application
spec:
  syncPolicy:
    syncOptions:
    - PrunePropagationPolicy=background
```

   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。

3. オプション: サイトを完全に削除する場合は、**SiteConfig** と、サイト固有の **PolicyGenerator** または **PolicyGentemplate** ファイルも Git リポジトリーから削除する必要があります。
4. オプション: サイトを再デプロイする場合など、一時的にサイトを削除する場合は、**SiteConfig** とサイト固有の **PolicyGenerator** または **PolicyGentemplate** CR を Git リポジトリーに残しておくことができます。

関連情報

- クラスターの削除の詳細は、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

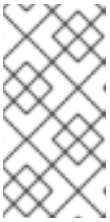
PolicyGenerator または **PolicyGentemplate** 設定の変更によってポリシーが古くなった場合 (ポリシーの名前を変更する場合など) は、次の手順を使用して古くなったポリシーを削除します。

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 リポジトリーに再び追加し、コミットしてリモートリポジトリーにプッシュします。



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. オプション: 別の方法として、**PolicyGenerator** または **PolicyGentemplate** CR に変更を加えたことでポリシーが古くなった場合は、これらのポリシーをハブクラスターから手動で削除できます。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 を使用したシングルノード 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、マネージドシングルノード 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** CR および **PolicyGenerator** CR に基づいたクラスターのサイトのインストールと設定のカスタムリソース (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 ディレクトリーには、**out/argocd/example/** フォルダー内の参照 **PolicyGenerator** 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-4.16"
  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 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. 前の手順の参照 **PolicyGenerator** CR を使用して、Day 2 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、シングルノード 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 **PolicyGenerator** 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
```

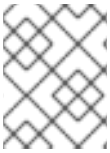
(1) カスタムラベルがノードに適用されます。

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

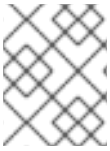
   (3) 関連する **SiteConfig** CR で設定された namespace と一致する必要があります

   (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 4.16 では、カーネル引数のみを追加できます。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。

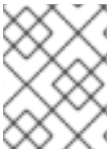
手順

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-4.16.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 Storage のデプロイ
- PolicyGenerator CR を使用した LVM Storage 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。

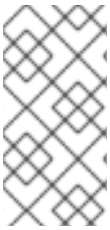
```
$ oc delete namespace <cluster_name>
```

   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、シングルノードクラスター、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。

CR	説明	使用法
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。
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 Secret** には、ターゲットのベアメタルホストにインストールされる OpenShift Container Platform イメージの認証情報が含まれています。
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シングルノード 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シングルノード 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 を使用したシングルノード 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群をデプロイするには、GitOps ZTP を使用したファーエッジサイトのデプロイ 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

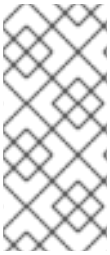
サブマイクロ秒の正確性を持つネットワーク内のノード間の同期を可能にします。

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

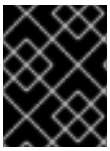
プロファイル	仮想 CPU	メモリー	ストレージ
最低限	4 - 8 個の仮想 CPU	32 GB のメモリー	120 GB



注記

1 つの仮想 CPU は 1 つの物理コアに相当します。ただし、同時マルチスレッディング (SMT) またはハイパースレッディングを有効にする場合は、次の式を使用して、1 つの物理コアを表す仮想 CPU の数を計算してください。

- (コアあたりのスレッド数 x コア数) x ソケット数 = 仮想 CPU



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。次の表は、Intel FlexRAN 4G および 5G baseband PHY リファレンスデザインに基づく、Intel Xeon Skylake サーバーおよびそれ以降のハードウェア世代の代表的なファームウェア設定を示しています。

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サンプル

ファームウェア設定	設定
CPU Power and Performance Policy	パフォーマンス
Uncore Frequency Scaling	無効
Performance P-limit	無効
Enhanced Intel SpeedStep ® Tech	有効
Intel Configurable TDP	有効
Configurable TDP Level	レベル 2
Intel® Turbo Boost Technology	有効

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

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**

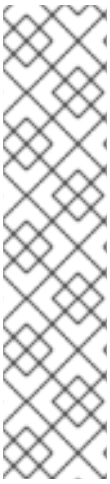
6.5. GITOPS ZTP を使用したシングルノード OPENSHIFT でのワークロードの分割

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異なる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。

あるいは、**SiteConfig** カスタムリソース (CR) の **cpuset** フィールドと、グループ **PolicyGenerator** または **PolicyGentemplate** CR の **reserved** フィールドを使用して、クラスター管理 CPU リソースを指定することもできます。GitOps ZTP パイプラインは、これらの値を使用して、シングルノード 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

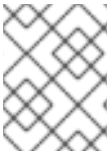
reserved CPU セットと **isolated** CPU セットが重複しないようにしてください。

関連情報

- 推奨されるシングルノード OpenShift ワークロードパーティショニング設定は、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。

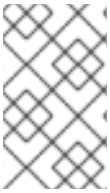
デフォルトで含まれる CR を変更するには、**SiteConfig** の **extraManifests** フィルターを使用します。詳細は、SiteConfig CR を使用した高度なマネージドクラスター設定 を参照してください。

6.6.1. ワークロードの分割

DU ワークロードを実行するシングルノード 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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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
aG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDAgIFVOSVQgW2VudmZpbGUgW3Zhcm5hbWVdXQogIGV
jaG8KICBlY2hvIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3RhcnQgc3Rhbn
phIGZyb20gdGhlIGdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuIGl0IHRvIHN0ZG91dAogIGVj
aG8KICBlY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZW
FkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGV
mYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUIGlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxC
n0KClVOSVQ9JDEKRU5WRklMRT0kMgpWQVJOQU1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUI
D09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCIgXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cm
FjdGluZyBFeGVjU3RhcnQgZnJvbSAkVU5JVCIKRklMRT0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCB
oZWFkIC1uIDEpCkZJTEU9JHtGSUxFI1wjIH0KaWYgW1sgISAtZiAkRklMRSBdXTsgdGhlbgogIGRlYnV
nICJGYWlsZWQgdG8gZmluZCByb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSIKICBle
Gl0CmZpCmRlYnVnICJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0k
```

```
KHNlZCAtbiAtZSAnL15FeGVjU3RhcnQ9LipcXCQvLC9bXlxcXSQvIHsgcy9eRXhlY1N0YXJ0PS8vOyBw
IH0nIC1lICcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHAgfScgJEZJTEUp
CgppZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUl
R9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogIGVja
G8gJEVYRUNTVEFSVApmaQo=
          mode: 493
          path: /usr/local/bin/extractExecStart
        - contents:
            source: data:text/plain;charset=utf-
8; base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbW\
zcGFjZS9tbnQgIiRAIgo=
          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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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
YnNvbHV0ZSBwb2QgY291bnQgKCsvLSkKIyAgNCUgLSBwZXJjZW50IGNoYW5nZSAoKy8tICAt
MSAtIGRpc2FibGUgdGhlIHN0ZWFkeS1zdGF0ZSBjaGVjawpTVEVBRFlfU1RBVEVfVEhSRVNIT0xEP
SR7U1RFQURZX1NUQVRFX1RlUkVTSE9MRDotMiV9CgojIERlZmF1bHQgc3RlYWR5LXN0YXRlIHd
pbmRvdyA9IDYwcwojIElmIHRoZSBydW5uaW5nIHBvZCBjb3VudCBzdGF5cyB3aXRoaW4gdGhlIGdpd
mVuIHRocmVzaG9sZCBmb3IgdGhpcyB0aW1lCiMgcGVyaW9kLCByZXR1cm4gQ1BVIHV0aWxpemF0
aW9uIHRvIG5vcm1hbCBiZWZvcmUgdGhlIG1heGltdW0gd2FpdCB0aW1lIGhhcwojIGV4cGlyZXMKU1
```

RFQURZX1NUQVRFX1dJTkRPVz0ke1NURUFEWV9TVEFURV9XSU5ET1c6LTYwfQoKIyBEZWZhd
Wx0IHN0ZWFkeS1zdGF0ZSBhbGxvd3MgYW55IHBvZCBjb3VudCB0byBiZSAic3RlYWR5IHN0YXRlIlgo
jIEluY3JlYXNpbmcgdGhpcyB3aWxsIHNraXAgYW55IHN0ZWFkeS1zdGF0ZSBjaGVja3MgdW50aWwgd
GhlIGNvdW50IHJpc2VzIGFib3ZlCiMgdGhpcyBudW1iZXIgdG8gYXZaWQgZmFsc2UgcG9zaXRpdmV
zIGlmIHRoZXJlIGFyZSBzb21lIHBlcmlvZHMgd2hlcmUgdGhlCiMgY291bnQgZG9lc24ndCBpbmNyZWF
zZSBidXQgd2Uga25vdyB3ZSBjYW4ndCBiZSBhdCBzdGVhZHktc3RhdGUgeWV0LgpTVEVBRFlfU1RB
VEVfTUlOSU1VTT0ke1NURUFEWV9TVEFURV9NSU5JTVVNOi0wfQoKIyMjIyMjIyMjIyMjIyMjIyMjIyMjI
yMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIwoKd2l0aGluKCkgewogIGxvY2FsIGxhc3Q9JDEgY3Vycm
udD0kMiB0aHJlc2hvbGQ9JDMKICBsb2NhbCBkZWx0YT0wIHBjaGFuZ2UKICBkZWx0YT0kCggY3Vy
cmVudCAtIGxhc3QgKSkKICBpZiBbWyAkY3VycmVudCAtZXEgJGxhc3QgXV07IHRoZW4KICAgIHBjaG
FuZ2U9MAogIGVsaWYgW1sgJGxhc3QgLWVxIDAgXV07IHRoZW4KICAgIHBjaGFuZ2U9MTAwMDA
wMAogIGVsc2UKICAgIHBjaGFuZ2U9JCgoICggIiRkZWx0YSIgKiAxMDApIC8gbGFzdCApKQogIGZpCi
AgZWNobyAtbiAibGFzdDokbGFzdCBjdCXJyZW50OiRjdXJyZW50IGRlbHRhOiRkZWx0YSBwY2hhbmdl
OiR7cGNoYW5nZX0lOiAiCkgewogIGxvY2FsIGxhc3Q9JDEgY3Vycm9udD0kMiBkZWx0YSBwY2hhbmdl
RlIgogIGxvY2FsIHQ9MCBzPTEwCiAgbG9jYWwgdG0gZXJpdENjb3VudD0wIGNjb3VudD0wIHN0ZWFkeVN
0YXRlVGltZT0wCiAgd2hpbGUgW1sgJHQgLWx0ICRNQVhJTVVNX1dBSVRfVElNRSBdXTsgZG8KIC
AgIHNsZWVwICRzCiAgICAoKHQgKz0gcykpCiAgICAjIERldGVjdCBzdGVhZHktc3RhdGUgcG9kIGNvd
W50CiAgICBjY291bnQ9JChjcmljdGwgcHMgMj4vZGV2L251bGwgfCB3YyAtbCkKICAgIGlmIFtbICRjY2
91bnQgLWd0IDAgXV0gJiYgc3RlYWR5c3RhdGUgIiRsYXN0Q2NvdW50IiAiJGNjb3VudCI7IHRoZW4KI
CAgICAgKChzdGVhZHlTdGF0ZVRpbWUgKz0gcykpCiAgICAgIGVjaG8gIlN0ZWFkeS1zdGF0ZSBmb3
IgJHtzdGVhZHlTdGF0ZVRpbWV9cy8ke1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgaWYgW1sgJHN0ZWFkeVN
1sgJHN0ZWFkeVN0YXRlVGltZSAtZ2UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICA
gICAgICBsb2dnZXIgIlJlY292ZXJ5OiBTdGVhZHktc3RhdGUgKCsvLSAkU1RFQURZX1NURUFEX1RI
UkVTSE9MRCkgZm9yICR7U1RFQURZX1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgICBicmVha
biAwCiAgICAgIGZpCiAgICBlbHNlCiAgICAgIGlmIFtbICRzdGVhZHlTdGF0ZVRpbWUgLWd0IDAgXV07
IHRoZW4KICAgICAgICBsb2dnZXIgIlJlY292ZXJ5OiBTdGVhZHktc3RhdGUgKC0vKyAkU1RFQURZX1RI
HRpbWVyIgogICAgICAgIGZpCiAgICBmaQogICAgbGFzdENjb3VudD0kY2NvdW50CiAgZG9uZQogIGxvZ2dlciAiUmVjb3Zlcnk6IFJlY292ZXJ5IENvbXBsZXRlIFRpbWVvdXQiCn0KCnNldFJjdU5vc
m1hbCgpIHsKICBlY2hvICJTZXR0aW5nIHJjdV9ub3JtYWwgdG8gMSIKICBlY2hvIDEgPiAvc3lzL2tlcm5lb
C9yY3Vfbm9ybWFsCn0KCm1haW4oKSB7CiAgd2FpdEZvclJlYWR5CiAgZWNobyAiV2FpdGluZyBmb3I
gc3RlYWR5IHN0YXRlIHRvb2s6ICQoYXdrICd7cHJpbnQgaW50KCQxLzM2MDApImgiLCBpbnQoKCQ
xJTM2MDApLzYwKSJtIiwgaW50KCQxJTYwKSJzIn0nIC9wcm9jL3VwdGltZSkiCiAgc2V0UmN1Tm9yb
WFsCn0KCmlmIFtbICIke0JBU0hfU09VUkNFWzBdfSIgPSAiJHswfSIgXV07IHRoZW4KICBtYWluICIke0
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ノード用に推奨される kdump 設定 (06-kdump-master.yaml)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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
```

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

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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)

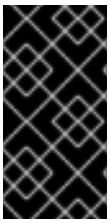
```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-crio-disable-wipe-master
spec:
  config:
    ignition:
      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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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
```

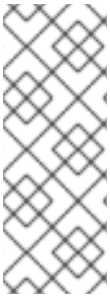
ワーカーノード用に推奨される ContainerRuntimeConfig (enable-crun-worker.yaml)

```
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、シングルノード 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シングルノード 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シングルノード 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:
    type: "vector"
```

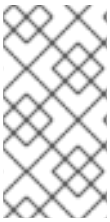
推奨されるログ転送設定 (ClusterLogForwarder.yaml)

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

#apiVersion: "logging.openshift.io/v1"
#kind: ClusterLogForwarder
#metadata:
#  name: instance
#  namespace: openshift-logging
#spec:
#  outputs:
#    - type: "kafka"
#      name: kafka-open
#      url: tcp://10.46.55.190:9092/test
#  pipelines:
#  - inputRefs:
#    - audit
#    - infrastructure
#    labels:
#      label1: test1
#      label2: test2
#      label3: test3
#      label4: test4
#    name: all-to-default
#    outputRefs:
#    - kafka-open
```

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シングルノード 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、Performance Addon 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。
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 **hugepages** が割り当てられた NUMA ノード (**node**) に設定します。

PerformanceProfile CR フィールド	説明
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)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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)

```
# Automatically generated by extra-manifests-builder
```

```
# Do not make changes directly.
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
            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
```

6.7.6. PTP

シングルノード 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: ordinary
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "ordinary"
      # 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: "ordinary"
        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)

```
# The grandmaster profile is provided for testing only
# It is not installed on production clusters
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 -w -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: 1500
          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
      - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,300
          - "-P"
          - "29.20"
          - "-p"
          - "CFG-MSG,1,38,300"
        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シングルノード 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: {}
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、シングルノード 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: {}
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
  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-related-kernel-args-master.yaml)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
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シングルノード 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

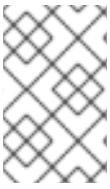
分散ユニットワークロードを実行するシングルノード OpenShift クラスターには、CPU リソースへの一貫したアクセスが必要です。Operator Lifecycle Manager (OLM) は定期的に Operator からパフォーマンスデータを収集するため、CPU 使用率が増加します。次の **ConfigMap** カスタムリソース (CR) は、OLM によるオペレーターパフォーマンスデータの収集を無効にします。

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 Storage 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: lvmcluster
  namespace: openshift-storage
  annotations: {}
spec: {}
#example: creating a vg1 volume group leveraging all available disks on the node
#         except the installation disk.
#  storage:
#    deviceClasses:
#    - name: vg1
#      thinPoolConfig:
#        name: thin-pool-1
#        sizePercent: 90
#        overprovisionRatio: 10
```

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 Storage 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 Storage 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シングルノード 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 を使用したファーエッジサイトのデプロイ

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

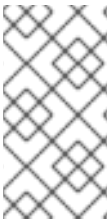
仮想化分散ユニット (vDU) アプリケーションをデプロイする前に、クラスターホストファームウェアおよびその他のさまざまなクラスター設定を調整および設定する必要があります。以下の情報を使用して、vDU ワークロードをサポートするためのクラスター設定を検証します。

関連情報

- GitOps ZTP を使用したシングルノード 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 4.16 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。



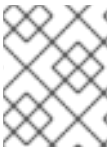
注記

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

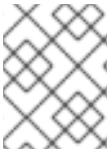
ファームウェア設定	設定	説明
HyperTransport (HT)	有効	HyperTransport (HT) バスは、AMD が開発したバス技術です。HT は、ホストメモリー内のコンポーネントと他のシステムペリフェラル間の高速リンクを提供します。
UEFI	有効	vDU ホストの UEFI からの起動を有効にします。
CPU Power and Performance Policy	パフォーマンス	CPU パワーとパフォーマンスポリシーを設定し、エネルギー効率よりもパフォーマンスを優先してシステムを最適化します。
Uncore Frequency Scaling	無効	Uncore Frequency 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。
Performance P-limit	無効	プロセッサーのアンコア周波数の調整を防ぐには、Performance P-limit を無効にします。

ファームウェア設定	設定	説明
Enhanced Intel® SpeedStep Tech	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。
Configurable TDP Level	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	無効	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	無効	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	無効	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。
Sub-NUMA Clustering	無効	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 4.16 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を説明します。

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

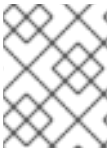
ztp-site-generate コンテナーから抽出した **MachineConfig** カスタムリソース (CR) がクラスターに適用されていることを確認します。CR は、抽出した **out/source-crs/extra-manifest/** フォルダーにあります。

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。
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。
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。

MachineConfig CR	説明
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 を使用したシングルノード 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 CPU と分離コア CPU の CPU 周波数を調整できます。FlexRAN のようなアプリケーションの場合、ハードウェアベンダーは、デフォルトで提供される周波数よりも低い CPU 周波数で実行することを推奨しています。周波数を設定する前に、プロセッサー世代の最大周波数設定に関する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。この例では、予約および分離された CPU の周波数を 2500 MHz に設定します。

```
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: 2500000
```

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 4.16 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を説明します。

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 OperatorHub 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。

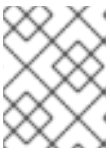
```
$ oc get catalogsource -A -o jsonpath='{range .items[*]}{.metadata.name}{" -- "}
{.metadata.annotations.target\.workload\.openshift\.io/management}{"\n"}{end}'
```

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

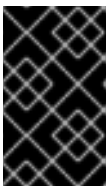
適切に設定された vDU クラスターでは、単一のアノテーション付きカタログソースのみがリスト表示されます。

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シングルノード 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 **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** アノテーションを追加し、変更を Git リポジトリーにプッシュします。

```
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. 次のコマンドを実行して、**BareMetalHost** オブジェクトに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 bmh -n <managed-cluster-namespace> <bmh-object> -ojsonpath='{.metadata}' | jq -r
'.annotations["bmac.agent-install.openshift.io/remove-agent-and-node-on-delete"]'
```

出力例

```
true
```

3. **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
# ...
```

4. 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

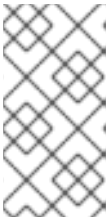
1. 次のコマンドを実行して、ワーカーノードの **BareMetalHost** および **Agent** CR がハブクラスターから削除されていることを確認します。

```
$ 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。

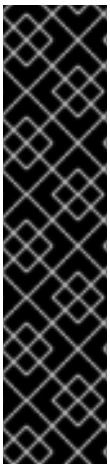
次のステップ

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 **Policy** 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 **PolicyGenerator** CR を使用して、適用される **Policy** 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です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

注記

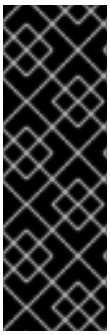
PolicyGenerator リソースの詳細は、RHACM の ポリシージェネレーターの統合 ドキュメントを参照してください。

9.1.1. RHACM PolicyGenerator と PolicyGenTemplate リソースのパッチ適用の比較

PolicyGenerator カスタムリソース (CR) と **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 を使用して OpenShift Container Platform リソースにパッチを適用する場合、**PolicyGenTemplate** CR よりも **PolicyGenerator** CR を使用する方が利点があります。RHACM **PolicyGenerator** API を使用すると、**PolicyGenTemplate** リソースでは不可能な、リソースにパッチを適用する一般的な方法が提供されます。

PolicyGenerator API は、Open Cluster Management 標準の一部ですが、**PolicyGenTemplate** API はその一部ではありません。次の表では、**PolicyGenerator** と **PolicyGenTemplate** リソースのパッチ適用および配置ストラテジーの比較を説明します。



重要

PolicyGenTemplate CR を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 と PolicyGenTemplate のパッチ適用の比較

PolicyGenerator のパッチ適用	PolicyGenTemplate のパッチ適用
リソースのマージには Kustomize ストラテジー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的管理 を参照してください。	変数をパッチで定義された値に置き換えることによって機能します。これは Kustomize マージストラテジーよりも柔軟性が低くなります。
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。
パッチ適用のみに依存し、埋め込み変数の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。
マージパッチ内のリストのマージはサポートされません。マージパッチ内のリストの置き換えはサポートされています。	リストのマージと置換は制限付きでサポートされており、リスト内の 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に OpenAPI 仕様 はサポートされていません。つまり、**PtpConfig** リソースなど、スキーマに従わないコンテンツをマージするには、パッチに追加のディレクティブが必要になります。	フィールドと値をパッチで定義された値に置き換えることによって機能します。
スキーマに従わないコンテンツをマージするには、パッチ内で **$patch: replace** などの追加のディレクティブが必要です。	ソース CR で定義されたフィールドと値を、パッチで定義された値 (例: **$name**) に置き換えます。
参照ソース CR で定義された **Name** と **Namespace** フィールドにパッチを適用できますが、CR ファイルにシングルオブジェクトがある場合に限られます。	参照ソース CR で定義された **Name** と **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

PolicyGenerator 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、クラスター設定に含めるカスタムリソース (CR)、生成されたポリシーに CR を統合する方法、およびオーバーレイコンテンツで更新する必要がある CR 内の項目を指示します。

次の例は、**ztp-site-generate** 参照コンテナーから抽出された **PolicyGenerator** CR (**acm-common-du-ranGen.yaml**) を示しています。**acm-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。

PolicyGenerator CR の例 - 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、非接続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ています。

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 CR を含めて構築できます。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 **PolicyGenerator** カスタムリソース (CR) をカスタマイズするときは、以下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 負荷が増加します。CR は **PolicyGenerator** CR の **policyName** フィールドに基づいてポリシーに統合されます。同じ **PolicyGenerator** 内で **policyName** の値が同じ CR は、シングルポリシーで管理されます。

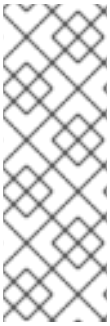
- 非接続環境では、すべての Operator を含む単一のインデックスとしてレジストリーを設定することにより、すべての Operator に対して単一のカタログソースを使用します。マネージドクラスターに **CatalogSource** CR を追加するたびに、CPU 使用率が増加します。

- **MachineConfig** CR は、インストール時に適用されるように **SiteConfig** CR に **extraManifests** 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、チャネルフィールドをオーバーライドして、目的のバージョンを明示的に識別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 デプロイメント用の PolicyGenerator CR

PolicyGenerator カスタムリソース (CR) を使用し、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。**PolicyGenerator** CR を使用すると、クラスター群の設定 CR のセットを管理するための 1 つ以上のポリシーを生成できます。**PolicyGenerator** CR は、マネージド CR のセットを識別し、それらをポリシーにバンドルして、それらの CR をラップするポリシーをビルドし、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 **PolicyGenerator** CR をベースに使用して、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 **PolicyGenerator** 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。

以下で、RAN クラスター設定に関連する **PolicyGenerator** CR を説明します。シングルノード、3 ノードのコンパクト、および標準のクラスター設定の違いに対応するために、グループ **PolicyGenerator** CR にバリアントが提供されています。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 デプロイメント用の 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	シングルノード 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。

PolicyGenerator CR	説明
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。
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	**PolicyGenerator** 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 **PolicyGenerator** CR。
acm-group-du-sno-validator-ranGen.yaml	シングルノード 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る **PolicyGenerator** CR。

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

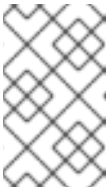
次の手順を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してプロビジョニングするマネージドクラスターに適用されるポリシーをカスタマイズします。

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. サイト固有の設定 CR 用の **PolicyGenerator** CR を作成します。
   a. **out/argocd/example/acmpolicygenerator/** フォルダーから CR に適した例 (**acm-example-sno-site.yaml** または **acm-example-multinode-site.yaml** など) を選択します。
   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 の **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ます。

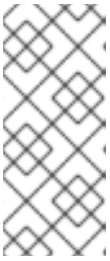
c. サンプルファイルの内容を目的の設定に合わせて変更します。

2. オプション: クラスター群全体に適用される一般的な設定 CR の **PolicyGenerator** CR を作成します。

   a. **out/argocd/example/acmpolicygenerator/** フォルダーから CR に適切な例 (**acm-common-ranGen.yaml** など) を選択します。

   b. 必要な設定に合わせてサンプルファイルの内容を変更します。

3. オプション: クラスター群の特定のグループに適用されるグループ設定 CR の **PolicyGenerator** 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。参考までに、**out/source-crs** ディレクトリーには、PolicyGenerator テンプレートで含めたりオーバーレイしたりできる source-crs の完全なリストが含まれています。



注記

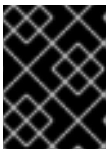
クラスターの特定の要件に応じて、クラスターのタイプごとに 1 つ以上のグループポリシーが必要になる場合があります。特に、サンプルのグループポリシーにはそれぞれ単一の **PerformancePolicy.yaml** ファイルがあり、それらのクラスターが同一のハードウェア設定で構成されている場合にのみ、クラスターのセット全体で共有できることを考慮すると、これが当てはまります。

   a. **out/argocd/example/acmpolicygenerator/** フォルダーから CR に適切な例 (**acm-group-du-sno-ranGen.yaml** など) を選択します。

   b. 必要な設定に合わせてサンプルファイルの内容を変更します。

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ー通知ポリシー **PolicyGenerator** CR を作成します。詳細は、「バリデーター通知ポリシーの作成」を参照してください。

5. サンプルの **out/argocd/example/acmpolicygenerator//ns.yaml** ファイルと同様の YAML ファイルですべてのポリシー namespace を定義します。



重要

Namespace CR を **PolicyGenerator** CR と同じファイルに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. **PolicyGenerator** 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。変更を **SiteConfig** CR と **PolicyGenerator** CR に同時にプッシュできます。

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 **PolicyGenerator** 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。アシストサービスによってマネージドクラスターに OpenShift Container Platform がインストールされた後、マネージドクラスターのポリシー同期の進行状況を監視できます。

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 annotations** 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

Policy カスタムリソース (CR) は、作成元の **PolicyGenerator** と同じ namespace に生成されます。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、**PolicyGenerator** 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-wrapped 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、以下次に示すエラーが生成されます。

```
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 **Placement** の **matchSelector** は、**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. **Placement** 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. 次のコマンドを使用して、どのポリシーが準拠しているかを確認します。

```
$ 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** ステータスで完了し、マネージドクラスターに **ztp-done** ラベルが適用されている場合は、**PolicyGenerator** を使用して追加の設定変更が可能な点に注意してください。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。

PolicyGenerator 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. **acm-group-du-sno-ranGen.yaml** ファイルで、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、「GitOps ZTP のトラブルシューティング」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。GitOps ZTP 完了状態への変更は、Red Hat Advanced Cluster Management (RHACM) バリデーター通知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です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

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。

GitOps ZTP パイプラインの完了後に追加設定を適用する

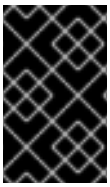
GitOps ZTP パイプラインのデプロイが完了すると、デプロイされたクラスターはアプリケーションのワークロードに対応できるようになります。この時点で、Operator を追加インストールし、お客様の要件に応じた設定を適用することができます。追加のコンフィギュレーションがプラットフォームのパフォーマンスや割り当てられた CPU バジェットに悪影響を与えないことを確認する。

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 を使用したソース CR コンテンツのオーバーライド

PolicyGenerator カスタムリソース (CR) を使用すると、**ztp-site-generate** コンテナー内の 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 への論理的なマージまたはパッチと考えることができます。**PolicyGenerator** CR を使用して、ベース CR のシングル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。

以下の手順の例では、**acm-group-du-sno-ranGen.yaml** ファイルの **PolicyGenerator** 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を説明します。この手順を元に、**PolicyGenerator** 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 **PolicyGenerator** CR にリストされているソース 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
   ```

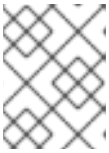
   b. ソース CR を抽出します。

   ```
   $ 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、**PolicyGenerator** 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** 参照ファイル内の **PerformanceProfile** の **PolicyGenerator** エントリーを更新します。以下の例の **PolicyGenerator** CR スタンザは、適切な CPU 仕様を指定し、**hugepages** を設定して、**globallyDisableIrqLoadBalancing** を false に設定する新しいフィールドを追加します。

```
- 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 **PolicyGenerator** 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。その CR の内容は、**PolicyGenerator** の **PerformanceProfile** エントリーの **metadata** および **spec** 内容をソース CR にマージすることによって生成されます。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 **PolicyGenerator** 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 から完全に省略されます。

例外として、**/source-crs** YAML ファイル内の **$mcp** 変数は、**PolicyGenerator** CR から **mcp** の指定値で代用されます。たとえば、**example/policygentemplates/acm-group-du-standard-ranGen.yaml** では、**mcp** の値は **worker** になります。

```
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. **PolicyGenerator** 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 **PolicyGenerator** 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 **/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めるように、**policies.manifests.path** を設定します。

4. Git で **PolicyGenerator** 変更をコミットしてから、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. 変更された **PolicyGenerator** を含めるように **ClusterGroupUpgrade** CR を更新し、**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、マネージド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。

PolicyGenerator カスタムリソース (CR) を使用して、デフォルトのポリシー評価間隔を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。

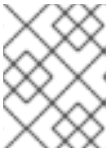
前提条件

- 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る。

手順

1. **PolicyGenerator** CR 内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** 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、**compliant** フィールドと **noncompliant** フィールドを **never** に設定して、特定の準拠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 内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. Git リポジトリー内の **PolicyGenerator** CR ファイルをコミットし、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. マネージドクラスターで **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、シングルノード 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 **PolicyGenerator** カスタムリソース (CR) を作成します。クラスタータイプごとにスタンドアロン **PolicyGenerator** CR が 1 つだけ必要です。たとえば、次の CR は、シングルノード OpenShift クラスターにバリデータ通知ポリシーを適用します。

シングルノードクラスターバリデーター通知ポリシー CR の例 (acm-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. Git リポジトリー内の **PolicyGenerator** CR ファイルをコミットし、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。

PolicyGenerator カスタムリソース (CR) を使用すると、**ztp-site-generate** コンテナー内の 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。

acm-group-du-sno-ranGen.yaml の **PolicyGenerator** 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。
- 「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。

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、**acm-group-du-sno-ranGen.yaml** の **PolicyGenerator** CR に基づき、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 を設定している。

手順

1. パフォーマンスモードを設定するには、**out/argocd/example/acmpolicygenerator//** の **acm-group-du-sno-ranGen.yaml** 参照ファイルで **PerformanceProfile** の **PolicyGenerator** エントリーを以下のように更新します。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        workloadHints:
          realTime: true
          highPowerConsumption: false
          perPodPowerManagement: false
```

2. Git で **PolicyGenerator** 変更をコミットしてから、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、**acm-group-du-sno-ranGen.yaml** の **PolicyGenerator** CR に基づき、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 を設定している。

手順

1. 高パフォーマンスモードを設定するには、**out/argocd/example/acmpolicygenerator/** の **acm-group-du-sno-ranGen.yaml** 参照ファイルで **PerformanceProfile** の **PolicyGenerator** エントリーを次のように更新します。

```
- path: source-crs/PerformanceProfile.yaml
  patches:
    - spec:
        workloadHints:
          realTime: true
          highPowerConsumption: true
          perPodPowerManagement: false
```

2. Git で **PolicyGenerator** 変更をコミットしてから、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、**acm-group-du-sno-ranGen.yaml** の **PolicyGenerator** CR に基づき、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 ステートを有効化している。

手順

1. 省電力モードを設定するには、**out/argocd/example/acmpolicygenerator/** の **acm-group-du-sno-ranGen.yaml** 参照ファイルで **PerformanceProfile** の **PolicyGenerator** エントリーを次のように更新します。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 **PolicyGenerator** 変更をコミットしてから、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。**acm-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の手順を説明します。

前提条件

- 「PolicyGenerator 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ている。

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、コアがすべて使用されているときにすべての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 **PolicyGenerator** 変更をコミットしてから、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 Storage 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 Storage 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の LVM Storage を設定するには、**acm-common-ranGen.yaml** ファイルの **policies.manifests** に次の YAML を追加します。

```
- 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 4.16 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、**acm-common-ranGen.yaml** ファイルを手動でオーバーライド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、**acm-common-ranGen.yaml** ファイルの **policies.manifests** に次の YAML を追加します。

```
- 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、**acm-group-du-sno-ranGen.yaml** ファイルに以下を追加します。

```
- 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 **PolicyGenerator** の変更をコミットしてから、その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 LVM Storage 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 トランスポートを使用する PTP イベントを設定できます。

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る。

手順

1. 要件に応じて、次の **PolicyGenerator** 変更を **acm-group-du-3node-ranGen.yaml**、**acm-group-du-sno-ranGen.yaml**、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、次の 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. 要件に応じて、**PtpConfigMaster.yaml** または **PtpConfigSlave.yaml** を指定できます。**acm-group-du-sno-ranGen.yaml** または **acm-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 を使用したソース CR コンテンツのオーバーライド

関連情報

- 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る。

手順

1. **acm-common-ranGen.yaml** ファイルの **policies.manifests** に次の YAML を追加して、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 (たとえば、**acm-group-du-sno-ranGen.yaml** ファイル) の **policies.manifests** に **HardwareEvent** CR を追加します。

```
- 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る。

手順

1. AMQ Interconnect Operator と Bare Metal Event Relay Operator を設定するには、**acm-common-ranGen.yaml** ファイルの **policies.manifests** に次の YAML を追加します。

```
# 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
```

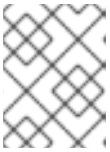
2. (**acm-example-sno-site.yaml** などの) サイト設定ファイルの **policies.manifests** に **Interconnect** CR を追加します。

```
- path: source-crs/AmqInstance.yaml
```

3. 特定のグループ設定ファイル (たとえば、**acm-group-du-sno-ranGen.yaml** ファイル) の **policies.manifests** に **HardwareEvent** CR を追加します。

```
- 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** で構成されます。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 **PolicyGenerator** 変更をコミットしてから、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、クラスターをネットワークのファーエッジにデプロイす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、**PolicyGenerator** 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 をインストールしている。

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. インストール中またはインストール後に、次のコマンドを実行して、ハブクラスターで **BareMetalHost** オブジェクトにアノテーションが表示されてい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、シングルノード 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 ノードでデバッグセッションを開始します。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、デバッグシェル内で **/host** をルート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。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. 次のコマンドを実行して、使用可能なすべてのブロックデバイスに関する情報をリスト表示します。

   ```
   # 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

PolicyGenerator (PGT) CR を使用して、イメージレジストリーを設定するために必要な CR を適用し、**imageregistry** 設定にパッチを適用します。

前提条件

- マネージド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 **PolicyGenerator** 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 **acm-example-sno-site.yaml** ファイルに追加します。

```
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 **PolicyGenerator** 変更をコミットしてから、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
```

- マネージドクラスターの **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
```

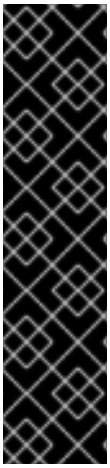
(1) **/var/imageregistry** は、ディスクが正しくパーティショニングされていることを示します。

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) を使用すると、GitOps Zero Touch Provisioning (ZTP) と 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です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

関連情報

- Topology Aware Lifecycle Manager の詳細は、Topology Aware Lifecycle Manager について 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。「OpenShift Container Platform イメージリポジトリーのミラーリング」(関連情報のリンクを参照) の手順に従って、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ます。**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。プラットフォームを更新するには、**PolicyGenerator** CR にイメージ署名を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。「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」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) の更新方法の詳細は、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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。
- 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ている。
- 目的のイメージリポジトリーをミラーリングしている。
- **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 RHACM ポリシーを作成している。

手順

1. プラットフォーム更新用の **PolicyGenerator** CR を作成します。
   a. 次の **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ための 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ています。イメージ署名は、プラットフォーム更新を適用する前にイメージを検証するために使用されます。

3 必要な 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非接続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. **PolicyGenerator** 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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。
- 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ている。
- 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 ポリシーを作成している。

手順

1. Operator 更新のために **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-upgrade PolicyGenerator** 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:
```

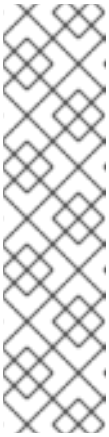
```
        - '*'
    evaluationInterval:
      compliant: 10m
      noncompliant: 10s
policies:
  - name: du-upgrade-operator-catsrc-policy
    policyAnnotations:
      ran.openshift.io/ztp-deploy-wave: "1"
    manifests:
      - path: source-crs/DefaultCatsrc.yaml
        patches:
          - metadata:
              name: redhat-operators-disconnected
            spec:
              displayName: Red Hat Operators Catalog
              image: registry.example.com:5000/olm/redhat-operators-disconnected: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) カタログ接続の監視状態を表示します。**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、**du-upgrade-operator-catsrc-policy** というポリシーが生成されます。これは、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-disconnected** カタログソースを更新するためのポリシーです。



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-disconnected** 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。

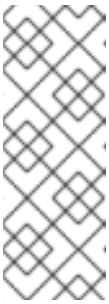
- 別のカタログソースの新しいインデックスイメージまたはレジストリーポーリング間隔の更新を使用して、別のカタログソースポリシーを準備します。
- 異なるカタログソースからの目的の Operator に対して個別のサブスクリプションポリシーを準備します。

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 **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 4.16 を通じて適用される Operator のデフォルトチャネルは、**performance-addon-operator** 以外は **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 **PolicyGenerator** 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 Operator 更新の失敗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 **PolicyGenerator** 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-disconnected
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators-disconnected:v{product-version}
        updateStrategy:
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY
- path: source-crs/DefaultCatsrc.yaml
  patches:
    - metadata:
        name: redhat-operators-disconnected-v2 (1)
      spec:
        displayName: Red Hat Operators Catalog v2 (2)
        image: registry.example.com:5000/olm/redhat-operators-disconnected:<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 **policies.manifests.patches.spec.image** フィールドは、**DefaultCatsrc.yaml** ファイル内のすべて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-disconnected-v2 (1)
# ...
```

1. **PolicyGenerator** 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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。
- 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 RHACM ポリシーを作成している。

手順

1. 「プラットフォーム更新の実」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 **PolicyGenerator** 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}'
```

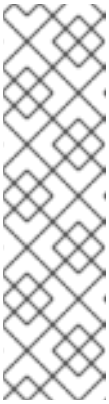
5. プラットフォームおよび Operator の更新を開始します。

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

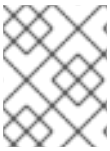
プラットフォームおよび Operator 更新の CR は、設定を **spec.enable: true** に設定して最初から作成できます。この場合、更新は事前キャッシュが完了した直後に開始し、CR を手動で有効にする必要はありません。

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 を使用したデプロイされたクラスターからの Performance Addon Operator サブスクリプションの削除

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。

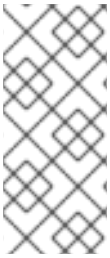
OpenShift Container Platform 4.11 以降を実行しているクラスターに Performance Addon Operator をインストールしないでください。OpenShift Container Platform 4.11 以降にアップグレードすると、Node Tuning Operator は Performance Addon Operator を自動的に削除します。



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** の 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、**acm-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、ArgoCD 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. **acm-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. PolicyGenerator CR を使用した Cluster Logging Operator アーティファクトの削除

バージョン 5.x から Cluster Logging Operator バージョン 6.x に更新する場合、新規 API カスタムリソース(CR)の作成後に、デプロイされたクラスターから古い Operator アーティファクトを削除するクリーンアップポリシーを追加する必要があります。

手順

1. **acm-group-du-clo5-cleanup.yaml** ファイルを、他の **PolicyGenerator** CR を含む git リポジトリーに追加します。

クリーンアップファイルの例

```
apiVersion: policy.open-cluster-management.io/v1
kind: PolicyGenerator
metadata:
  name: group-du-sno-clo5-cleanup
placementBindingDefaults:
  name: group-du-sno-clo5-cleanup-placement-binding
policyDefaults:
  namespace: ztp-group
  # Allow this policy to be unbound from clusters based on a custom label
  placement:
    labelSelector:
      matchExpressions:
      - key: group-du-sno
        operator: Exists
      - key: du-profile
        operator: In
        values: ["latest"]
      - key: clo5-cleanup-done
        operator: DoesNotExist
  remediationAction: inform
  severity: low
  # standards: []
  namespaceSelector:
    exclude:
      - kube-*
    include:
      - '*'
  evaluationInterval:
```

```
    compliant: 10m
    noncompliant: 10s
policies:
- name: group-du-sno-clo5-cleanup
  policyAnnotations:
    ran.openshift.io/ztp-deploy-wave: "11" (1)
  manifests:
    - path: source-crs/ClusterLogging5Cleanup.yaml
```

(1) デフォルト値は **443** です。**group-du-sno-clo5-cleanup** ポリシーの wave は、デフォルトで **10** の **group-du-sno** 設定ポリシーの後にあることを確認してください。

2. クラスターに関連付けられたフリートで使用されているラベルと一致するように、必要に応じて **acm-group-du-clo5-cleanup.yaml** ファイルのバインディングルールを更新します。
3. **acm-group-du-clo5-cleanup.yaml** 行を **kustomization.yaml** ファイルの **generators** セクションに追加します。

   kustomization.yaml ファイルの例

```
generators:
- acm-common-ranGen.yaml
# ...
- acm-group-du-clo5-cleanup.yaml
# ...
```

4. 変更をコミットしてから、git リポジトリーにプッシュします。
5. ArgoCD が生成され、クリーンアップポリシーが生成され、ハブクラスターに同期するまで待ちます。

検証

1. クリーンアップポリシーが、Cluster Logging Operator バージョン 5.x から 6.x にアップグレードしたすべてのクラスターにバインドされていることを確認します。
2. クリーンアップポリシーの wave が **group-du-sno** 設定ポリシーの後にいることを確認します。

関連情報

Logging 6.0 へのアップグレード

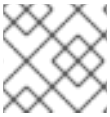
9.3.8. シングルノード OpenShift クラスター上の TALM を使用したユーザー指定のイメージの事前キャッシュ

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シングルノード 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。

9.3.8.1. 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

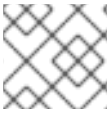
クラスターにイメージをインストールすると、それらを変更したり削除したりすることはできません。

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 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@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1
```

```
ef
    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@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1ef
      -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47adfaef
      -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga0991234afdadfsa09
    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. イメージがシングルノード 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の詳細は、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。

9.3.9. 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復し、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 **Policy** 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た **Policy** CR を生成する方法をカスタマイズできます。

重要

PolicyGenTemplate CR を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較

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、**spec.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。

PolicyGenTemplate CR の例 - 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 負荷が増加します。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。

- 非接続環境では、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 **extraManifests** 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。

- **PolicyGenTemplate** CR は、チャネルフィールドをオーバーライドして、目的のバージョンを明示的に識別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 common、group、および site 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す。

10.1.3. RAN デプロイメントの PolicyGenTemplate CR

PolicyGenTemplate カスタムリソース (CR) を使用し、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、クラスター群の設定 CR のセットを管理するための 1 つ以上のポリシーを生成できます。**PolicyGenTemplate** CR は、マネージド CR のセットを識別し、それらをポリシーにバンドルして、それらの CR をラップするポリシーをビルドし、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、シングルノード、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

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	シングルノード 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、シングルノード 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。

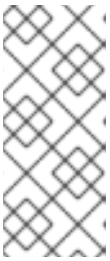
手順

1. 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 **spec.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 spec.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ます。

   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. 必要な設定に合わせてサンプルファイルの内容を変更します。
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。参考までに、**out/source-crs** ディレクトリーには、PolicyGenTemplate テンプレートで含めたりオーバーレイしたりできる source-crs の完全なリストが含まれています。



注記

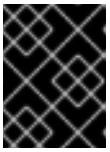
クラスターの特定の要件に応じて、クラスターのタイプごとに 1 つ以上のグループポリシーが必要になる場合があります。特に、サンプルのグループポリシーにはそれぞれ単一の **PerformancePolicy.yaml** ファイルがあり、それらのクラスターが同一のハードウェア設定で構成されている場合にのみ、クラスターのセット全体で共有できることを考慮すると、これが当てはまります。

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. 必要な設定に合わせてサンプルファイルの内容を変更します。

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、「バリデーター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/policygentemplateskustomization.yaml** で示す例と同様に、**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。アシストサービスによってマネージドクラスターに OpenShift Container Platform がインストールされた後、マネージドクラスターのポリシー同期の進行状況を監視できます。

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 annotations** 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

Policy カスタムリソース (CR) は、作成元の **PolicyGenTemplate** と同じ namespace に生成されます。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-wrapped 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、以下次に示すエラーが生成されます。

```
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. 次のコマンドを使用して、どのポリシーが準拠しているかを確認します。

```
$ 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** ステータスで完了し、マネージドクラスターに **ztp-done** ラベルが適用されている場合は、**PolicyGenTemplate** を使用して追加の設定変更が可能な点に注意してください。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、「GitOps ZTP のトラブルシューティング」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

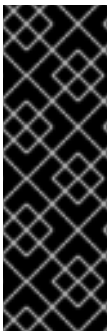
クラスターのインストールと設定は、GitOps ZTP 完了フェーズで実行されます。これは、**ztp-running** ラベルを削除し、**ManagedCluster** CR に **ztp-done** ラベルを追加することで表示されます。**ztp-done** ラベルは、設定が適用され、ベースライン DU 設定が完了したことを示しています。GitOps ZTP 完了状態への変更は、Red Hat Advanced Cluster Management (RHACM) バリデーター通知ポリシーの準拠状態が条件となります。このポリシーは、完了したインストールの既存の基準をキャプチャし、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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較

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。

GitOps 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を説明します。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
   ```

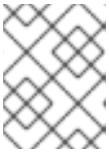
   b. ソース CR を抽出します。

   ```
   $ 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生成されます。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。たとえば、**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、マネージド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る。

手順

1. **PolicyGenTemplate** CR 内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、**compliant** フィールドと **noncompliant** フィールドを **never** に設定して、特定の準拠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 内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。

```
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. マネージドクラスターで **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、シングルノード 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、シングルノード OpenShift クラスターにバリデータ通知ポリシーを適用します。

シングル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. **{policy-gen-crs}** オブジェクトの名前。この名前は、**placementBinding**、**placementRule**、および要求された **namespace** で作成される **policy** の一部としても使用されます。
2. この値は、**policy-gen-crs** グループで使用される **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、シングルノードの場合は **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。シングル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. PolicyGenTemplate 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。
- 「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。

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 を設定している。

手順

1. パフォーマンスモードを設定するには、**out/argocd/example/policygentemplates//** の **group-du-sno-ranGen.yaml** 参照ファイルで **PerformanceProfile** の **PolicyGenTemplate** エントリーを次のように更新します。

```
- 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 を設定している。

手順

1. 高パフォーマンスモードを設定するには、**out/argocd/example/policygentemplates/** の **group-du-sno-ranGen.yaml** 参照ファイル内で **PerformanceProfile** の **PolicyGenTemplate** エントリーを次のように更新します。

```
- 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 ステートを有効化している。

手順

1. 省電力モードを設定するには、**out/argocd/example/policygentemplates/** の **group-du-sno-ranGen.yaml** 参照ファイル内で **PerformanceProfile** の **PolicyGenTemplate** エントリーを次のように更新します。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。

```
- 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を説明します。

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ている。

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 Storage 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 Storage 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。

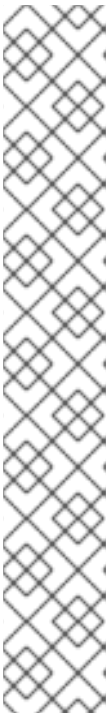
前提条件

- 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 Storage 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 4.16 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 Storage 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 トランスポートを使用する PTP イベントを設定できます。

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る。

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. **spec.sourceFiles** で、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、次の 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. 要件に応じて、**PtpConfigMaster.yaml** または **PtpConfigSlave.yaml** を指定できます。**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

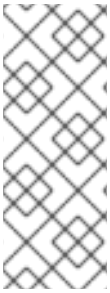
- PolicyGenTemplate CR を使用して、ソース CR の内容を上書きする。

関連情報

- 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る。

手順

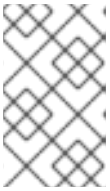
1. 次の YAML を **common-ranGen.yaml** ファイルの **spec.sourceFiles** に追加して、Bare Metal Event Relay Operator を設定します。

```
# 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る。

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 (**example-sno-site.yaml** ファイルなど) の**spec.sourceFiles** 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** で構成されます。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、クラスターをネットワークのファーエッジにデプロイす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 をインストールしている。

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. インストール中またはインストール後に、次のコマンドを実行して、ハブクラスターで **BareMetalHost** オブジェクトにアノテーションが表示されてい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、シングルノード 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 ノードでデバッグセッションを開始します。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、デバッグシェル内で **/host** をルート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。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. 次のコマンドを実行して、使用可能なすべてのブロックデバイスに関する情報をリスト表示します。

   ```
   # 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

- マネージド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
```

- マネージドクラスターの **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) を使用すると、GitOps Zero Touch Provisioning (ZTP) と 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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較
- 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。「OpenShift Container Platform イメージリポジトリーのミラーリング」(関連情報のリンクを参照) の手順に従って、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ます。**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。「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」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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。

- 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ている。
- 目的のイメージリポジトリーをミラーリングしている。
- **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 RHACM ポリシーを作成している。

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非接続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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。
- 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ている。

- 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 ポリシーを作成している。

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-upgrade PolicyGenTemplate** 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-disconnected
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators-disconnected:v4.16 (1)
        updateStrategy: (2)
          registryPoll:
            interval: 1h
      status:
        connectionState:
          lastObservedState: READY (3)
```

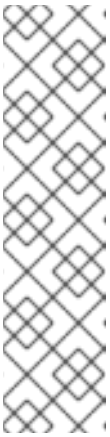
(1) インデックスイメージ URL には、必要な Operator イメージが含まれます。インデックスイメージが常に同じイメージ名とタグにプッシュされている場合、この変更は必要ありません。

(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。この動作に対処するには、更新が完了したら **registryPoll.interval** をデフォルト値に戻します。

(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、**du-upgrade-operator-catsrc-policy** というポリシーが生成されます。これは、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-disconnected** カタログソースを更新するためのポリシーです。



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-disconnected** 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 4.16 を通じて適用される Operator のデフォルトチャネルは、**performance-addon-operator** 以外は **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 Operator 更新の失敗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-disconnected
    spec:
      displayName: Red Hat Operators Catalog
      image: registry.example.com:5000/olm/redhat-operators-disconnected:v{product-
version}
      updateStrategy:
        registryPoll:
          interval: 1h
    status:
      connectionState:
        lastObservedState: READY
- fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "operator-catsrc-policy"
    metadata:
      name: redhat-operators-disconnected-v2 (1)
    spec:
      displayName: Red Hat Operators Catalog v2 (2)
      image: registry.example.com:5000/olm/redhat-operators-disconnected:<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-disconnected-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) をインストールしている。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新している。
- 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 RHACM ポリシーを作成している。

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 を使用したデプロイされたクラスターからの Performance Addon Operator サブスクリプションの削除

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、ArgoCD 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. PolicyGenTemplate CR を使用した Cluster Logging Operator アーティファクトの削除

バージョン 5.x から Cluster Logging Operator バージョン 6.x に更新する場合、新規 API カスタムリソース(CR)の作成後に、デプロイされたクラスターから古い Operator アーティファクトを削除するクリーンアップポリシーを追加する必要があります。

手順

1. 他の **PolicyGenTemplate** CR を含む git リポジトリーに **group-du-clo5-cleanup-policy.yaml** ファイルを追加します。

   クリーンアップファイルの例

```
---
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  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/description: ""
    policy.open-cluster-management.io/standards: NIST SP 800-53
    ran.openshift.io/ztp-deploy-wave: "11"
  name: group-du-sno-clo5-cleanup
  namespace: ztp-group
spec:
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: group-du-sno-clo5-cleanup
        spec:
          evaluationInterval:
            compliant: 10m
            noncompliant: 10s
          namespaceSelector:
            exclude:
              - kube-*
            include:
              - '*'
          object-templates-raw: |
            {{ if ne (default "" (lookup "apiextensions.k8s.io/v1" "CustomResourceDefinition"
"" "clusterlogforwarders.logging.openshift.io").metadata.name) "" }}
            - complianceType: mustnothave
```

```
                  objectDefinition:
                    apiVersion: logging.openshift.io/v1
                    kind: ClusterLogForwarder
                    metadata:
                      name: instance
                      namespace: openshift-logging
                - complianceType: mustnothave
                  objectDefinition:
                    apiVersion: apiextensions.k8s.io/v1
                    kind: CustomResourceDefinition
                    metadata:
                      name: clusterlogforwarders.logging.openshift.io
                {{ end }}
                {{ if ne (default "" (lookup "apiextensions.k8s.io/v1" "CustomResourceDefinition"
"" "clusterloggings.logging.openshift.io").metadata.name) "" }}
                - complianceType: mustnothave
                  objectDefinition:
                    apiVersion: logging.openshift.io/v1
                    kind: ClusterLogging
                    metadata:
                      name: instance
                      namespace: openshift-logging
                - complianceType: mustnothave
                  objectDefinition:
                    apiVersion: apiextensions.k8s.io/v1
                    kind: CustomResourceDefinition
                    metadata:
                      name: clusterloggings.logging.openshift.io
                {{ end }}
              remediationAction: inform
              severity: low
    remediationAction: inform
---
apiVersion: apps.open-cluster-management.io/v1
kind: PlacementRule
metadata:
    name: placementrule-group-du-sno-clo5-cleanup
    namespace: ztp-group
spec:
    clusterSelector:
        matchExpressions:
            - key: group-du-sno
              operator: Exists
            - key: du-profile
              operator: In
              values:
                - latest
            - key: clo5-cleanup-done
              operator: DoesNotExist
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
    name: binding-group-du-sno-clo5-cleanup
    namespace: ztp-group
placementRef:
```

```
    apiGroup: apps.open-cluster-management.io
    kind: PlacementRule
    name: placementrule-group-du-sno-clo5-cleanup
subjects:
    - apiGroup: policy.open-cluster-management.io
      kind: Policy
      name: group-du-sno-clo5-cleanup
```

2. 必要に応じて、クリーンアップ **PolicyGenTemplate** CR の **PlacementRule** を更新して、クラスターの関連するフリートで使用されているバインディングルールと一致するようにします。

3. クリーンアップ **PolicyGenTemplate** CR を **kustomization.yaml** ファイルの **resources** セクションに追加します。

   kustomization.yaml ファイルの例

```
# ...

resources:
- ns.yaml
# ...
- group-du-clo5-cleanup-policy.yaml
# ...
```

4. クリーンアップ **PolicyGenTemplate** CR および **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。

5. ArgoCD が生成され、クリーンアップポリシーが生成され、ハブクラスターに同期するまで待ちます。

検証

1. クリーンアップポリシーが、Cluster Logging Operator バージョン 5.x から 6.x にアップグレードしたすべてのクラスターにバインドされていることを確認します。

2. クリーンアップポリシーの wave が **group-du-sno** 設定ポリシーの後にいることを確認します。

関連情報

- Logging 6.0 へのアップグレード

10.3.8. 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シングルノード 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。

10.3.8.1. 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 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@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1
ef
      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@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d38
08fc925ada29c559a47e2e1ef
        -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d38
08fc925ada29c559a47adfaef
        -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafd
edga0991234afdadfsa09
      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. イメージがシングルノード 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

10.3.9. 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復し、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
```

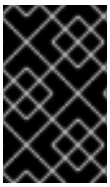
```
- group-spoke1-config-policy
- spoke1-config-policy
- group-spoke1-validator-du-policy
preCaching: false
remediationStrategy:
  maxConcurrency: 1
  timeout: 240
```

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

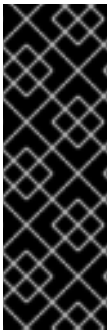
Topology Aware Lifecycle Manager は、GitOps Zero Touch Provisioning (ZTP) で使用される設定ポリシーで、Red Hat Advanced Cluster Management (RHACM) ハブクラスターテンプレート機能をサポートします。

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 を使用してポリシーを管理し、マネージドクラスターにデプロイすることは、OpenShift Container Platform の今後のリリースでは非推奨になります。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 を使用する場合に利用できます。

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 リソースのパッチ適用の比較
- 設定ポリシーでのテンプレート処理に対する RHACM のサポート

11.1. グループ POLICYGENERATOR または POLICYGENTEMPLATE CR でのグループとサイト設定の指定

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。**PolicyGenerator** または **PolicyGentemplate** CR のサイト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 で管理する場合は、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ために必要なグループポリシー CR は 1 つだけです。

次の例は、3 つの **ConfigMap** CR と 1 つの **PolicyGenerator** CR を使用して、ハードウェアタイプとリージョン別にグループ化されたクラスターに、サイト設定とグループ設定の両方を適用する方法を示しています。

注記

ConfigMap CR には、1 MiB のサイズ制限 (Kubernetes ドキュメント) があります。**ConfigMap** CR の実際のサイズは、**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
```

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る。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、 **PolicyGenerator** CR グループから生成されるポリシーと同じ namespace に存在する必要があります。

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 **PolicyGenerator** または **PolicyGentemplate** CR グループを作成します。

a. グループ **PolicyGenerator** CR を作成します。この例の **PolicyGenerator** CR は、**policyDefaults.placement** フィールド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: 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. **PolicyGenerator** または **PolicyGentemplate** CR のサイトを Git でコミットし、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。

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 **PolicyGenerator** 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。

11.2. 新しい 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。

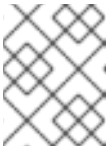
手順

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です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

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
- クラスターに修復されたポリシーのセット
- クラスターに修復されたポリシーの順序
- 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、帯域幅が制限されたクラスター用にイメージの事前キャッシュを提供します。

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) から自動的に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

- 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 をインストールしている。
- TALM 4.16 には RHACM 2.9 以降が必要。
- 非接続のレジストリーでハブクラスターをセットアップしている。
- **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. 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 をインストールしている。
- TALM 4.16 には RHACM 2.9 以降が必要。
- 非接続の registry 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. **Subscription** CR 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復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。

詳細は、「マネージド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構成されます。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。

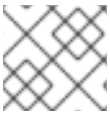
有効な field が false に設定されたサンプル ClusterGroupUpgrade CR

```
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: ""
  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復が完了したときに 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. 修復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 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. 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。

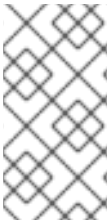
TALM は **Progressing** 条件を使用して、ステータスと理由を次のように報告します。

- **true**
  TALM は準拠していないポリシーを修復しています。

- **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がすべての管理ポリシーに準拠している。
  - 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た。
  - CR のブロックがシステムにないか、まだ完了していない。

- **ClusterGroupUpgrade** CR が有効になっていない。



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: {}
  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。

12.5.5. 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**
  修復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復に時間がかかりすぎたため、ポリシーの修復に失敗しました。
  - 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。

timedout 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: {}
  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、ポリシーの修復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。

12.5.6. 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) をインストールしている。
- 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 ポリシーを作成している。

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** カスタムリソース (CR) で指定されたクラスターの **inform** ポリシーのセットを修復します。TALM は、**PlacementBinding** CR の **bindingOverrides.remediationAction** および **subFilter** 仕様を通じて **Policy** CR の **remediationAction** 仕様を制御することで、**inform** ポリシーを修復します。各ポリシーには、対応する独自の 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 がバッチで更新を完了すると、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復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、クラスターにそれ以上の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) をインストールしている。
- TALM 4.16 には RHACM 2.9 以降が必要。
- 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 ポリシーを作成している。

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
```

- 現在のバッチからクラスターに適用された子ポリシーの場合、**spec.remediationAction** の値は **enforce** に変更されます。
- 残りのクラスターの子ポリシーの場合、**spec.remedationAction** の値は **inform** のままとなります。
- バッチが完了すると、適用された子ポリシーの場合、**spec.remediationAction** の値は **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シングル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シングル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. シングル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シングル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復している場合、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。

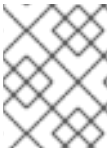
```
$ oc get csv -n <operator_namespace>
```

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. 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 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、ポリシーの修復を開始できます。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。

12.7.1. 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。

12.7.2. 事前キャッシュでの ClusterGroupUpgrade CR の作成

シングルノード 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) をインストールしている。
- 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。

12.8. 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

12.8.1. 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。

12.8.2. 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>
```

12.8.3. 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
```

12.8.4. 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、マネージドクラスターの少なくとも 1 つと一致するか確認します。

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
```

12.8.5. 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
```

12.8.6. 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 **PolicyGenerator** または **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** namespace 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。
- 非接続環境では、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。

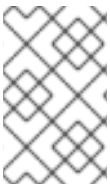
   ```
   $ oc describe job -n openshift-talo-pre-cache pre-cache
   ```

関連情報

- OpenShift Container Platform の Operator の問題のトラブルシューティング
- Topology Aware Lifecycle Manager による管理ポリシーの更新
- PolicyGenerator CRD について

第13章 GITOPS ZTP を使用したシングルノード 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。シングルノード OpenShift クラスターにワーカーノードを追加すると、元のシングルノード 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、既存のシングルノード OpenShift クラスターのダウンタイムは必要ありません。

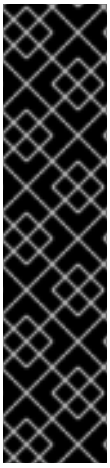


注記

シングルノード 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復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** イグニッション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復してから、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は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

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シングルノード OpenShift クラスターの詳細は、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 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) の common、group、および site 固有の **PolicyGenerator** リソースまたは **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** コンテナーを抽出した場合に、関連する **out/argocd/example** フォルダーで見つけることができる次の CR が含まれています。

/acmpolicygenerator リソース

- **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,"enableOperatorWebh
ook":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 同期が失われ、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 を使用したワーカーノードポリシーのワーカーノードへの適用

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,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMyIgfQo=
                      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復するには、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,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMyIgfQo=
              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,ewogICJtYW5hZ2VtZW50Ijogewog ICAgImNwdXNldCI6ICIwLTMiCiAgfQp9Cg==
              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復するには、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 を使用してシングルノード 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シングルノード 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シングルノード 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.Embedded.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

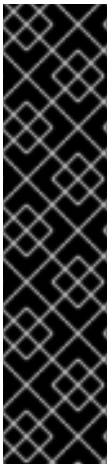
```
example-sno [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/master":"","node-role.kubernetes.io/worker":""}
example-node2 [{"status":"True","type":"Ready"}] {"node-role.kubernetes.io/worker":""}
```

第14章 シングルノード OPENSHIFT デプロイメント用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。リモートのシングルノード 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 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、以下のリンクを参照してください。

- テクノロジープレビュー機能のサポート範囲

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。非接続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/VirtualMedia.Ins
ertMedia
```

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 の作成に関する詳細は、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 ツールの使用に関する詳細は、Integrated Dell Remote Access Controller 9 RACADM CLI Guide を参照してください。
- HP HPONCFG ツールの使用に関する詳細は、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

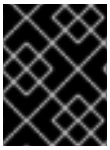
```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0     7:0    0  93.8G  0 loop /run/ephemeral
loop1     7:1    0 897.3M  1 loop /sysroot
sr0      11:0    1   999M  0 rom  /run/media/iso
nvme0n1 259:1    0   1.5T  0 disk
```

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
```

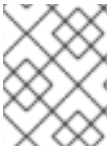
```
sr0         11:0    1   999M  0 rom  /run/media/iso
nvme0n1     259:1    0   1.5T  0 disk
└─nvme0n1p1 259:2    0   250G  0 part /var/mnt
```
(1)

(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、シングルノード 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_edec37e7cd8b1611d0031d45e7958361c65e2005f145b471a8108f1b54316c07.t
gz
-rw-r--r--. 1 root root  156092894 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_ee51b062b9c3c9f4fe77bd5b3cc9a3b12355d040119a1434425a824f137c61a9.tg
z
-rw-r--r--. 1 root root  172297800 Oct 31 15:29 ocp-v4.0-art-
dev@sha256_ef23d9057c367a36e4a5c4877d23ee097a731e1186ed28a26c8d21501cd82718.t
gz
-rw-r--r--. 1 root root  171539614 Oct 31 15:23 ocp-v4.0-art-
dev@sha256_f0497bb63ef6834a619d4208be9da459510df697596b891c0c633da144dbb025.t
gz
-rw-r--r--. 1 root root  160399150 Oct 31 15:20 ocp-v4.0-art-
dev@sha256_f0c339da117cde44c9aae8d0bd054bceb6f19fdb191928f6912a703182330ac2.tgz

-rw-r--r--. 1 root root  175962005 Oct 31 15:17 ocp-v4.0-art-
dev@sha256_f19dd2e80fb41ef31d62bb8c08b339c50d193fdb10fc39cc15b353cbbfeb9b24.tgz

-rw-r--r--. 1 root root  174942008 Oct 31 15:33 ocp-v4.0-art-
dev@sha256_f1dbb81fa1aa724e96dd2b296b855ff52a565fbef003d08030d63590ae6454df.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_f48b68d5960ba903a0d018a10544ae08db5802e21c2fa5615a14fc58b1c1657c.tg
z
-rw-r--r--. 1 root root  168899617 Oct 31 15:16 ocp-v4.0-art-
dev@sha256_f5099b0989120a8d08a963601214b5c5cb23417a707a8624b7eb52ab788a7f75.t
gz
```

```
-rw-r--r--. 1 root root  176592362 Oct 31 15:05 ocp-v4.0-art-dev@sha256_f68c0e6f5e17b0b0f7ab2d4c39559ea89f900751e64b97cb42311a478338d9c3.tgz
-rw-r--r--. 1 root root  157937478 Oct 31 15:37 ocp-v4.0-art-dev@sha256_f7ba33a6a9db9cfc4b0ab0f368569e19b9fa08f4c01a0d5f6a243d61ab781bd8.tgz

-rw-r--r--. 1 root root  145535253 Oct 31 15:26 ocp-v4.0-art-dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de5.tgz
-rw-r--r--. 1 root root  158048761 Oct 31 15:40 ocp-v4.0-art-dev@sha256_f914228ddbb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb47.tgz

-rw-r--r--. 1 root root  167914526 Oct 31 15:37 ocp-v4.0-art-dev@sha256_fa3ca9401c7a9efda0502240aeb8d3ae2d239d38890454f17fe5158b62305010.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_b86b8aea29a818a9c22944fd18243fa0347c7a2bf1ad8864113ff2bb2d8e0726.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_7753a8d9dd5974be8c90649aadd7c914a3d8a1f1e016774c7ac7c9422e9f9958
Processing artifact [2/379]: ose-kube-rbac-proxy@sha256_c27a7c01e5968aff16b6bb6670423f992d1a1de1a16e7e260d12908d3322431c

Processing artifact [3/379]: ocp-v4.0-art-dev@sha256_370e47a14c798ca3f8707a38b28cfc28114f492bb35fe1112e55d1eb51022c99
...
Processing artifact [378/379]: ose-local-storage-operator@sha256_0c81c2b79f79307305e51ce9d3837657cf9ba5866194e464b4d1b299f85034d0
Processing artifact [379/379]: multicluster-operators-channel-rhel8@sha256_c10f6bbb84fe36e05816e873a72188018856ad6aac6cc16271a1b3966f73ceb3

...
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の詳細は、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=precache-
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%3A-
ai-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER%0
A%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%20tot
al%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%20%
27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%20%7
C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%20%24ur
i%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%20existin
g%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7Buri%7
D%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%2
0%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%20copy%20
dir%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_RELEASE_LI
ST%7D%0A%0A%23%20workaround%20while%20https%3A%2F%2Fgithub.com%2Fopenshift%2Fa
ssisted-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%20workaround%2
0for%20bugzilla%201964591%20where%20symlinks%20inside%20%2Fvar%2Flib%2Fcontainers%2F
%20get%0A%23%20corrupted%20under%20some%20circumstances.%0A%23%0A%23%20In%20or
der%20to%20let%20agent.service%20start%20correctly%20we%20are%20checking%20here%20whe
ther%20the%20requested%0A%23%20container%20image%20exists%20and%20in%20case%20%2
2podman%20images%22%20returns%20an%20error%20we%20try%20removing%20the%20faulty%
0A%23%20image.%0A%23%0A%23%20In%20such%20a%20scenario%20agent.service%20will%20d
etect%20the%20image%20is%20not%20present%20and%20pull%20it%20again.%20In%20case%0A
%23%20the%20image%20is%20present%20and%20can%20be%20detected%20correctly%2C%20no
%20any%20action%20is%20required.%0A%0AIMAGE%3D%24%28echo%20%241%20%7C%20sed
%20%27s%2F%3A.%2A%2F%2F%27%29%0Apodman%20image%20exists%20%24IMAGE%20%7
C%7C%20echo%20%22already%20loaded%22%20%7C%7C%20echo%20%22need%20to%20be%
20pulled%22%0A%23podman%20images%20%7C%20grep%20%24IMAGE%20%7C%7C%20podm
an%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=precache-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\nExecStart=b
ash /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%3A-
ocp-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER%0
A%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%20tot
al%0Acurrent_copy%3D1%0A%0Awhile%20read%20-
r%20line%3B%0Ado%0A%20%20uri%3D%24%28echo%20%22%24line%22%20%7C%20awk%20%
27%7Bprint%241%7D%27%29%0A%20%20%23tar%3D%24%28echo%20%22%24line%22%20%7
C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%20%24ur
i%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%20existin
g%20image%20%24tar%22%0A%20%20%20%20%20%20echo%20%22Copying%20%24%7Buri%7
D%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%2
0%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%20copy%20
dir%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_RELEASE_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。

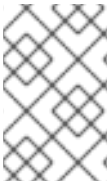
追加のパラメーターは、ライブ RHCOS をディスクに書き込む **coreos-installer** ユーティリティーに直接渡されます。次回の再起動時に、オペレーティングシステムはディスクから起動します。

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。

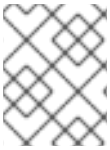
extract-ocp.sh

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して GitOps ZTP ワークフローを開始できます。

14.6. "RENDERED CATALOG IS INVALID" エラー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 以降では、Lifecycle Agent によって、シングルノード 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別の方法が提供されます。イメージベース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、OpenShift Container Platform <4.y> から <4.y+2>、<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 クラスターに新しい **ostree** stateroot としてインストールされている専用のシードクラスターから生成された OCI イメージを使用します。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の設定を使用してデプロイされたシングルノード 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任意のシングルノード OpenShift クラスター上の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 Agent は、参加しているクラスター上の 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行います。

- シードクラスターでは、**SeedGenerator** CR によってシードイメージの生成が可能になります。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 によって、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が指定されます。

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 の目的のステージを定義します。値は、**Idle**、**Prep**、**Upgrade**、または **Rollback** になります。

(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、およびイメージにアクセスするため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、指定された時間枠内にアップグレードが完了しなかった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) (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含む **ConfigMap** リソースのリストを指定します。

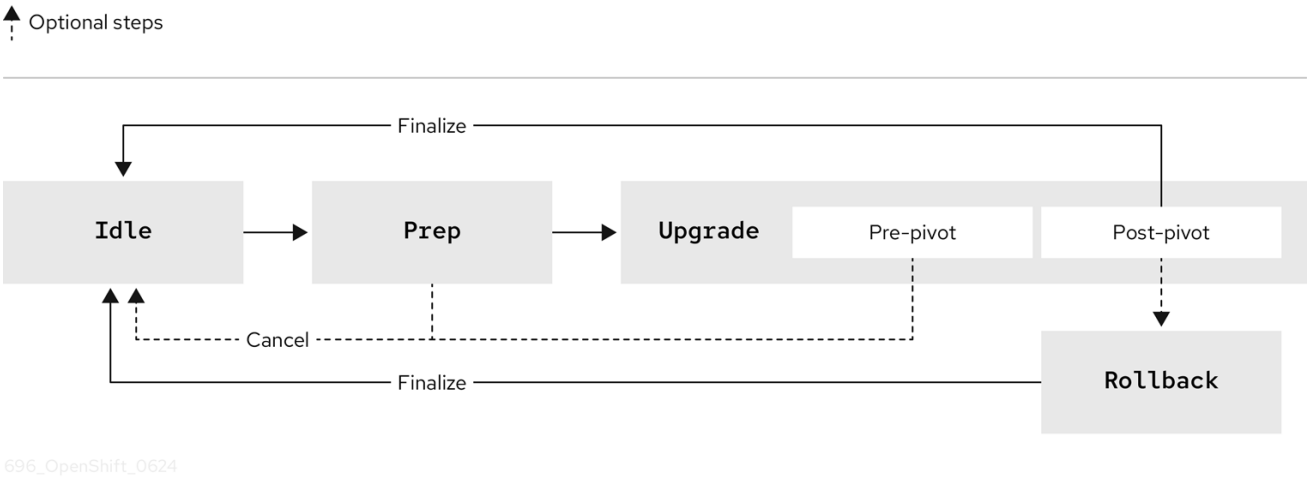
(5) OADP **Backup** および **Restore** CR を含む **ConfigMap** リソースのリストを指定します。

15.1.1. イメージベースアップグレードの各ステージ

シードクラスターでシードイメージを生成した後、**ImageBasedUpgrade** CR で **spec.stage** フィールドを以下のいずれかの値に設定することで、ターゲットクラスターのステージを移動できます。

- **Idle**
- **Prep**
- **Upgrade**
- **Rollback** (オプション)

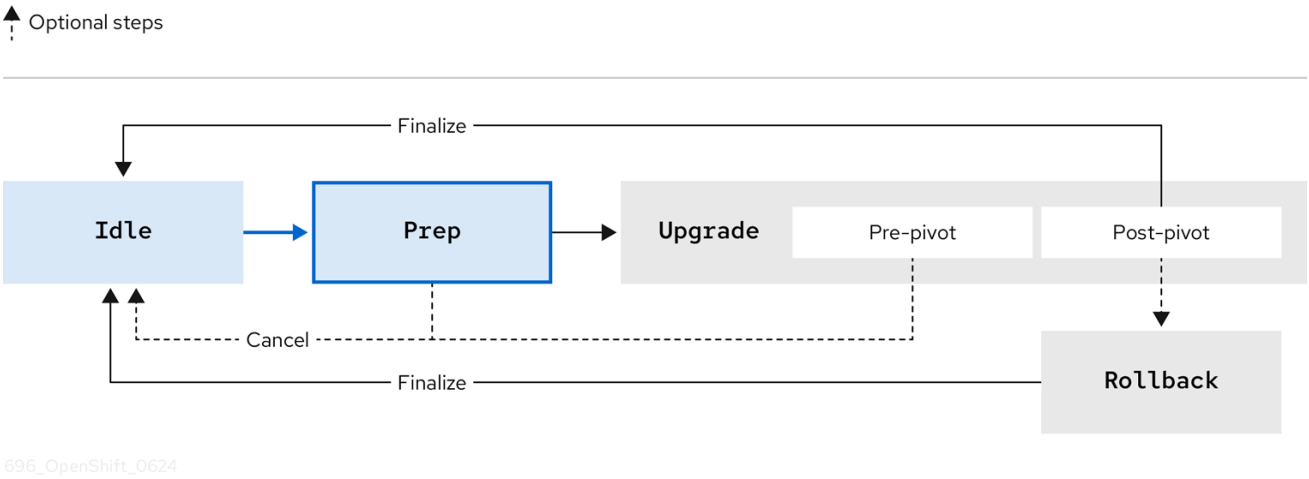
図15.1 イメージベースアップグレードの各ステージ



15.1.1.1. Idle ステージ

Lifecycle Agent は、Operator が最初にデプロイされるときに、**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テージです。進行中のアップグレードはなく、クラスターは **Prep** ステージに移行する準備ができています。

図15.2 Idle ステージからの移行

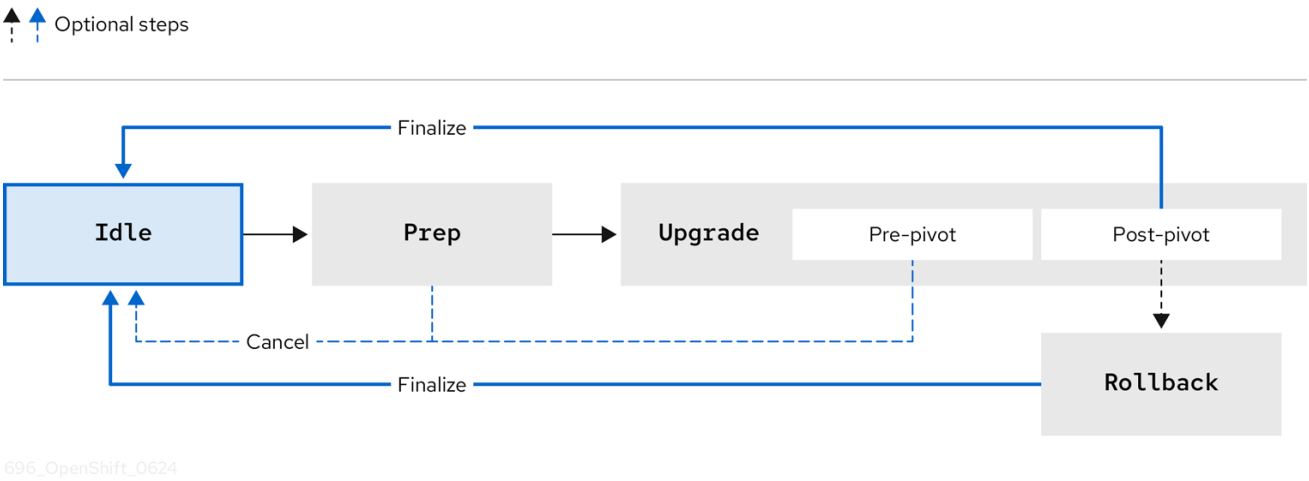


Idle ステージに移行して、次のいずれかの手順も実行します。

- 正常なアップグレードを終了する
- ロールバックを終了する
- **Upgrade** ステージのピボット前フェーズまで、進行中のアップグレードをキャンセルする

Idle ステージに移行すると、Lifecycle Agent によってリソースが確実にクリーンアップされるため、クラスターの再アップグレードへの準備が整います。

図15.3 Idle ステージへの移行



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

このステージは、スケジュールされたメンテナンス期間の前に完了できます。

Prep ステージでは、**ImageBasedUpgrade** CR で以下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
- 適用する追加のマニフェストと、アップグレード後に保持するカスタムカタログソース (ある場合)

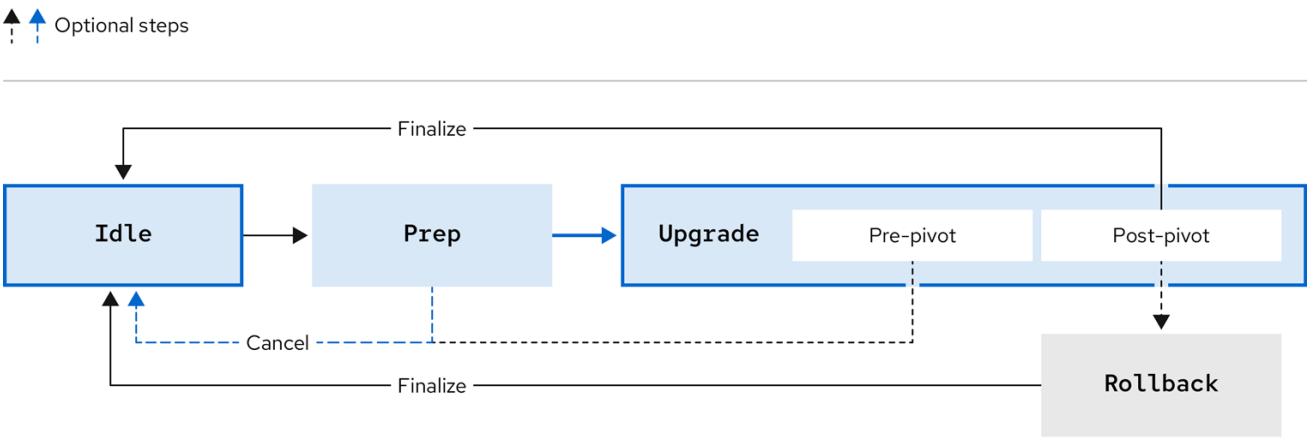
次に、指定した内容に基づいて、Lifecycle Agent は現在実行中のバージョンに影響を与えずにアップグレードの準備をします。このステージでは、Lifecycle Agent は、ターゲットクラスターが特定の条件を満たしているかどうかをチェックして、**Upgrade** ステージに進む準備ができていることを確認します。Operator により、シードイメージが、シードイメージで指定された追加のコンテナーイメージとともに、ターゲットクラスターにプルされます。Lifecycle Agent は、コンテナーストレージディスクに十分な領域があるかどうかを確認します。必要に応じて、ディスク使用量が指定のしきい値を下回るまで、固定されていないイメージが Operator によって削除されます。コンテナーストレージディスクのクリーンアップを設定または無効にする方法の詳細は、「コンテナーストレージディスクの自動イメージクリーンアップの設定」を参照してください。

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、クラスターを再設定し、クラスターを RHACM に登録して、アプリケーションアーティファクトを復元するために **Upgrade** ステージで使用されます。

OADP Operator に加えて、Lifecycle Agent は **ostree** バージョン管理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の後に、クラスターの完全な再設定が可能となります。

Prep ステージが終了したら、**Idle** ステージに移動して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。

図15.4 Prep ステージからの移行



15.1.1.3. Upgrade ステージ

Upgrade ステージは 2 つのフェーズで構成されています。

ピボット前

新しい stateroot にピボットする直前に、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** ステージ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上に作成されます。Lifecycle Agent は、**ImageBasedUpgrade** CR の **extraManifests** フィールドに指定された CR、またはターゲットクラスターにバインドされた ZTP ポリシーに記述されている CR をエクスポートします。ピボット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。

ピボット後

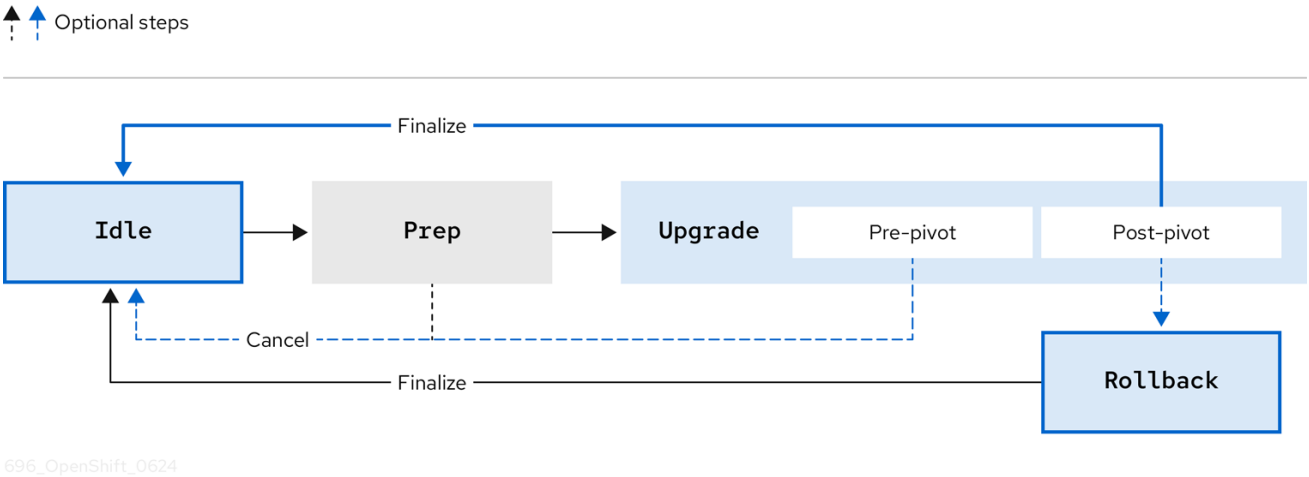
新しい stateroot から起動した後、Lifecycle Agent はシードイメージのクラスター暗号化も再生成します。これにより、同じシードイメージでアップグレードされた各シングルノード OpenShift クラスターに、一意で有効な暗号化オブジェクトが確保されます。次に、Operator は、ピボット前のフェーズで収集されたクラスター固有のアーティファクトを適用して、クラスターを再設定します。Operator は保存されたすべての CR を適用し、バックアップを復元します。

アップグレードが完了し、変更が正常に行われたら、**Idle** ステージに移行してアップグレードを終了できます。

重要

アップグレードを終了すると、元のリリースにロールバックすることはできません。

図15.5 Upgrade ステージからの移行



アップグレードをキャンセルする場合は、**Upgrade** ステージのピボット前フェーズまではキャンセルできます。アップグレード後に問題が発生した場合は、**Rollback** ステージに進んで手動でロールバックすることができます。

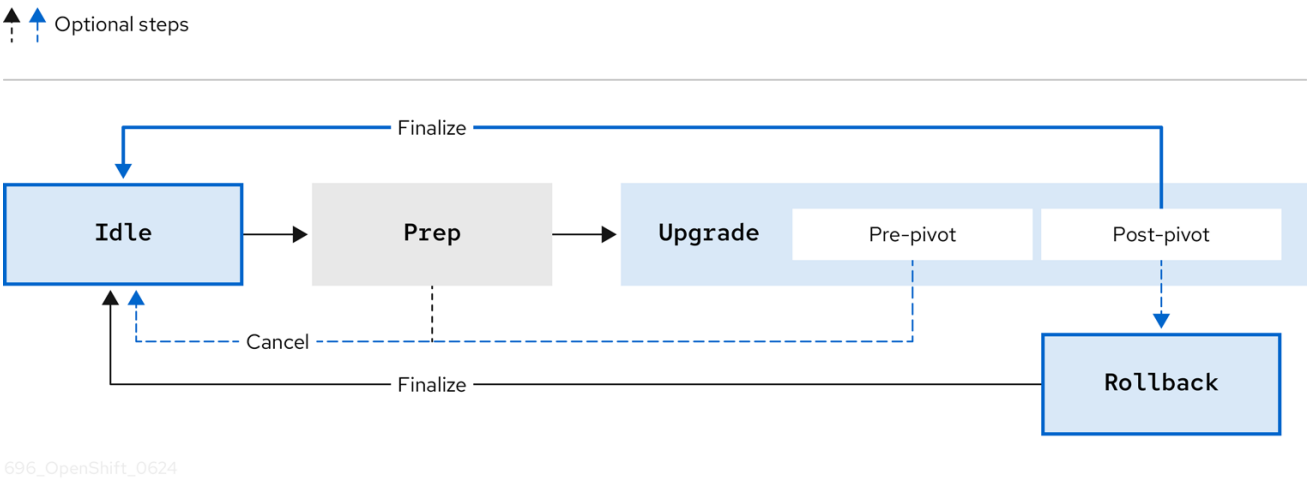
15.1.1.4. Rollback ステージ

Rollback ステージは、失敗時に手動または自動で開始できます。**Rollback** ステージでは、Lifecycle Agent 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。その後、ノードは OpenShift Container Platform の以前のリリースと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** ステージに移行すると、Lifecycle Agent 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。

指定された制限時間内にアップグレードが完了しない場合、Lifecycle Agent は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、「Lifecycle Agent を使用した Rollback ステージへの移行」または「Lifecycle Agent と GitOps ZTP を使用した Rollback ステージへの移行」セクションを参照してください。

図15.6 Rollback ステージからの移行



関連情報

- コンテナーストレージディスクの自動イメージクリーンアップの設定
- Lifecycle Agent を使用したシングルノード OpenShift クラスターのイメージベースアップグレードの実行
- GitOps ZTP を使用したシングルノード OpenShift クラスターのイメージベースアップグレードの実行

15.1.2. イメージベースアップグレードのガイドライン

イメージベースアップグレードを成功させるには、デプロイメントが特定の要件を満たしている必要があります。

イメージベースアップグレードを実行できるさまざまなデプロイメント方法があります。

GitOps ZTP

クラスターをデプロイおよび設定するには、GitOps Zero Touch Provisioning (ZTP) を使用します。

非 GitOps

クラスターを手動でデプロイおよび設定します。

非接続環境でイメージベースアップグレードを実行できます。非接続環境でイメージをミラーリングする方法の詳細は、「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

デプロイメント方法に応じて、イメージベースアップグレードには次の最小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	1.4.1	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	いいえ
RHACM	2.10.2	いいえ
GitOps ZTP プラグイン	4.16	GitOps ZTP デプロイメント方式のみ
Red Hat OpenShift GitOps	1.12	GitOps ZTP デプロイメント方式のみ
Topology Aware Lifecycle Manager (TALM)	4.16	GitOps ZTP デプロイメント方式のみ
Local Storage Operator [1]	4.14	はい
論理ボリュームマネージャー (LVM) ストレージ [1]	4.14.2	はい

1. 永続ストレージは、LVM Storage または Local Storage Operator のいずれかによって提供される必要があり、両方によって提供される必要はありません。

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、ハブクラスターは次の条件を満たしている必要があります。

- シードイメージに RHACM リソースが含まれないようにするには、シードイメージを生成する前に、オプションの RHACM アドオンをすべて無効にする必要があります。
- ターゲットのシングルノード OpenShift クラスターでイメージベースアップグレードを実行する前に、ハブクラスターを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、同じハードウェアと同様の設定を持つシングルノードの OpenShift クラスターのセットを対象とします。つまり、シードクラスターは、次の項目についてターゲットクラスターの設定と一致する必要があります。

- CPU トポロジー

- CPU コア数
- チューニング済みのパフォーマンス設定 (予約 CPU 数など)

- ターゲットクラスターの **MachineConfig** リソース
- IP バージョン

注記

この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されていません。

- Lifecycle Agent と OADP Operator を含む Day 2 Operator のセット
- 非接続レジストリー
- FIPS 設定

次の設定は、参加するクラスターで部分的に一致している必要があります。

- 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も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。
- 参加するすべてのクラスターでは、コンテナーストレージ用のプライマリーディスク上に専用のパーティションが必要です。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 内の **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の作成方法の詳細は、「ostree stateroot 間の共有コンテナーパーティションの設定」または「GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定」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は、「シードイメージの設定」および「RAN DU プロファイルを使用したシードイメージの設定」を参照してください。

関連情報

- ostree stateroot 間の共有コンテナーパーティションの設定
- GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定
- シードイメージの設定

15.1.2.4. OADP バックアップおよび復元ガイドライン

OADP Operator を使用すると、**ConfigMap** オブジェクトにラップされた **Backup** および **Restore** CR を使用して、ターゲットクラスター上の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、OpenShift Container Platform の現行バージョンとターゲット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が含まれている必要があります。

次のリソースは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** (Local Storage Operator (LSO) を使用する場合)

シングルノード 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)

Lifecycle Agent は、**localvolume** リソースとそれに関連付けられた **StorageClass** リソースを含む、必要なアーティファクトを自動的にバックアップおよび復元します。アプリケーション **Backup** CR で **persistentvolumes** リソースを除外する必要があります。

LVM Storage

LVM Storage アーティファクトの **Backup** 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 **Backup** CR に **persistentVolumes** リソースを含める必要があります。

イメージベースアップグレードの場合、特定のターゲットクラスターでは 1 つの Operator のみがサポートされます。

重要

両方の Operator の場合、**ImageBasedUpgrade** CR を介して Operator CR を追加のマニフェストとして適用しないでください。

永続ボリュームの内容は保存され、ピボット後も使用されます。**DataProtectionApplication** CR を設定するときは、イメージベースアップグレード用に **.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** or **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数値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と、アノテーションの値に基づいて昇順に適用されます。アノテーションを定義しない場合は、それらが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 (RHACM や LVM Storage アーティファクトなど) では、アプリケーションのアノテーションよりも数値的に低くする必要があります。これにより、アプリケーションの前にプラットフォームアーティファクトが復元されます。

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ている場合は、バックアップの範囲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設定するために、個別の **Backup** CR と **Restore** CR を作成する必要があります。クラススコープのリソースの **Restore** CR は、残りのアプリケーションの **Restore** CR の前に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したリソースに基づいて、Lifecycle Agent は、**Backup** CR の作成時に、指定されたリソースに **lca.openshift.io/backup: <backup_name>** ラベルを適用し、**labelSelector.matchLabels.lca.openshift.io/backup: <backup_name>** ラベルセレクターを追加します。

特定のリソースをバックアップするために **lca.openshift.io/apply-label** アノテーションを使用するには、アノテーションにリストされているリソースも **spec** セクションに含める必要があります。**lca.openshift.io/apply-label** アノテーションが **Backup** CR で使用される場合、**spec** セクションで他のリソースタイプが指定されているかどうかに関係なく、アノテーションにリストされてい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、クラススコープのリソースの場合は **group/version/resource/name** 形式、namespace スコープのリソースの場合は **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 にアタッチされている必要があります。

15.1.2.5. 追加のマニフェストガイドライン

Lifecycle Agent は、新しい stateroot デプロイメントで再起動した後、そして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が異なると、追加のマニフェストを適用する方法も異なります。

GitOps ZTP

lca.openshift.io/target-ocp-version: <target_ocp_version> ラベルを使用して、ピボット後に Lifecycle Agent が抽出および適用する必要がある追加の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** のラベルが付けられたマニフェストの数を指定できます。指定されている場合、Lifecycle Agent は、Prep ステージと Upgrade ステージで、ポリシーから抽出されたマニフェストの数が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
```

非 Gitops

適用順序を決定するには、追加のマニフェストを **lca.openshift.io/apply-wave** アノテーションでマークします。ラベル付けされた追加マニフェストは **ConfigMap** オブジェクトにラップされ、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指す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

以下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 サブスクリプション

関連情報

- Lifecycle Agent を使用したシングルノード OpenShift クラスターのイメージベース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

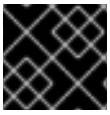
15.2. シングルノード OPENSHIFT クラスターのイメージベースアップグレードの準備

15.2.1. イメージベースアップグレード用の共有コンテナーパーティションの設定

イメージベースアップグレードを行うには、シングルノード OpenShift クラスターに共有の **/var/lib/containers** パーティションを設定する必要があります。これはインストール時に行うことができます。

15.2.1.1. ostree stateroot 間の共有コンテナーパーティションの設定

インストール時にシードクラスターとターゲットクラスターの両方に **MachineConfig** を適用して、個別のパーティションを作成し、アップグレードプロセス中に使用される 2 つの **ostree** stateroot 間で **/var/lib/containers** パーティションを共有します。



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** を適用して個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-path/pci-<root_disk> 1
          partitions:
            - label: var-lib-containers
              startMiB: <start_of_partition> 2
              sizeMiB: <partition_size> 3
      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
            [Unit]
            Before=local-fs.target
            Requires=systemd-fsck@dev-disk-by\x2dpartlabel-var\x2dlib\x2dcontainers.service
            After=systemd-fsck@dev-disk-by\x2dpartlabel-var\x2dlib\x2dcontainers.service

            [Mount]
            Where=/var/lib/containers
            What=/dev/disk/by-partlabel/var-lib-containers
            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. 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するために、パーティションの最小サイズを 500 GB に指定します。値が小さすぎると、インストール後のデプロイメントが失敗します。

15.2.1.2. GitOps ZTP を使用する場合の ostree stateroot 間の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** パーティションを共有します。

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 をインストールしている。詳細は、「Butane のインストール」を参照してください。

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-<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 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するために、パーティションの最小サイズを 500 GB に指定します。値が小さすぎると、インストール後のデプロイメントが失敗します。

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-0000:00:17.0-ata-1.0","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. 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:
        - hostName: <name>
          ignitionConfigOverride: '{"ignition":{"version":"3.2.0"},"storage":{"disks":[{"device":"/dev/disk/by-path/pci-0000:00:17.0-ata-1.0","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"}]}}'
[...]
```

検証

1. インストール中またはインストール後に、次のコマンドを実行して、ハブクラスターで **BareMetalHost** オブジェクトにアノテーションが表示されてい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-path/pci-0000:00:17.0-ata-1.0\",\"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\"}]}}"
```

2. インストール後、次のコマンドを実行して、シングルノード 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
```

関連情報

- Butane のインストール

15.2.2. イメージベース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、アップグレードに向けてクラスターを準備します。

GitOps 以外の方法で 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
- Backup CR の作成
- Restore CR の作成

15.2.2.1. CLI を使用した Lifecycle Agent 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 (例: **lcao-namespace.yaml**) を作成します。

```
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 (例: **lcao-operatorgroup.yaml**) を作成します。

```
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/v1alpha1
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 リソースを調べます。

```
$ 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、Lifecycle Agent が起動して実行されてい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 Lifecycle Agent 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 のリストから **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-lifecycle-agent** 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. Lifecycle Agent が、**InstallSucceeded ステータス** で **openshift-lifecycle-agent** プロジェクトにリスト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** をクリックし、**Operator Subscriptions** タブおよび **Install Plans** タブの **ステータス** で障害やエラーがないか検査します。
2. **Workloads → Pods** をクリックし、**openshift-lifecycle-agent** プロジェクト内の Pod のログを確認します。

15.2.2.3. GitOps ZTP を使用した Lifecycle Agent のインストール

イメージベースアップグレードを実行するには、GitOps Zero Touch Provisioning (ZTP) を使用して Lifecycle Agent をインストールします。

手順

1. **ztp-site-generate** コンテナーイメージから次の CR を抽出し、**source-cr** 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/v1alpha1
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
│   ├── 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: LcaSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: LcaSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: 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。

手順

1. **ztp-site-generate** コンテナーイメージから次の CR を抽出し、**source-cr** ディレクトリーにプッシュします。

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/v1alpha1
kind: Subscription
metadata:
  name: redhat-oadp-operator
  namespace: openshift-adp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "2"
spec:
  channel: stable-1.4
  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
|    ├── 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: OadpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: OadpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: OadpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: OadpOperatorStatus.yaml
      policyName: "subscriptions-policy"
[...]
```

3. ターゲットクラスター専用の **DataProtectionApplication** CR と S3 シークレットを作成します。

   a. **ztp-site-generate** コンテナーイメージから次の CR を抽出し、**source-cr** ディレクトリーにプッシュします。

   DataProtectionApplication.yaml ファイルの例

```
apiVersion: oadp.openshift.io/v1alpha1
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、バケット内に自動的に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、それらの間の干渉を避けるために、各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。各ターゲットクラスターに一意のストレージディレクトリーを確保するには、RHACM ハブテンプレート関数を使用できます (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の場所の可用性を確認します。

b. オーバーライドを使用して、サイトの **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: OadpSecret.yaml
    policyName: "config-policy"
    data:
      cloud: <your_credentials> 1
  - fileName: 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: 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 **bucket** 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。**prefix** は、**bucket** 内に自動的に作成されるサブディレクトリーの名前を定義します。**bucket** と **prefix** の組み合わせは、それらの間の干渉を避けるために、各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。各ターゲットクラスターに一意のストレージディレクトリーを確保するには、RHACM ハブテンプレート関数を使用できます (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、**SeedGenerator** 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. シードイメージの設定

シードイメージは、同じハードウェアと同様の設定を持つシングルノードの OpenShift クラスターのセットを対象とします。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含まれている必要があります。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。

次の表は、シードイメージに含めるべき、および含めるべきでないコンポーネント、リソース、設定を示しています。

表15.2 シードイメージの設定

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 バージョン [1]	はい
Lifecycle Agent と OADP Operator を含む Day 2 Operator のセット	はい
切断されたレジストリー設定 [2]	はい
有効なプロキシー設定 [3]	はい
FIPS 設定	はい
ターゲットクラスターのサイズに一致する、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 の **LocalVolume** で使用される **StorageClass** • LSO の **LocalVolume** • LVMS の **LVMCluster** CR	いいえ
OADP **DataProtectionApplication** CR	いいえ

1. この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されていません。
2. シードクラスターが非接続環境にインストールされている場合は、ターゲットクラスターも非接続環境にインストールする必要があります。
3. 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の設定

次の表は、RAN DU プロファイルを使用する際にシードイメージに含めるべき、および含めるべきでないコンポーネント、リソース、設定を示しています。

表15.3 RAN DU プロファイルを使用したシードイメージの設定

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
DisableOLMPprof.yaml	はい
TunedPerformancePatch.yaml	はい
PerformanceProfile.yaml	はい
SriovOperatorConfig.yaml	はい
DisableSnoNetworkDiag.yaml	はい
StorageClass.yaml	x (**StorageLV.yaml** で使用されている場合)
StorageLV.yaml	いいえ
StorageLVMCluster.yaml	いいえ

表15.4 追加のマニフェスト用の 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

リソース	追加マニフェストとして適用
ClusterLogForwarder.yaml	はい **注記** DU プロファイルには Cluster Logging Operator が含まれますが、このプロファイルでは Cluster Logging Operator CR は設定または適用されません。ログ転送を有効にするには、追加のマニフェストとして **ClusterLogForwarder.yaml** CR を含めてください。追加のマニフェストは、イメージベースのアップグレードプロセス中に、ターゲットのシングルノード OpenShift クラスターに適用されます。
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と共通である場合は、それらをシードイメージに含めることができます。それ以外の場合は、追加のマニフェストとして適用します。
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まったく同じである場合は、それらをシードイメージに含めることができます。それ以外の場合は、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、**SeedGenerator** 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て、イメージ生成を開始します。シードイメージの生成には、次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
- シードイメージの生成および **SeedGenerator** CR で指定されたイメージリポジトリーへのシードイメージのプッシュ
- クラスター Operator の復元

- 期限切れのシードクラスター証明書
- シードクラスター用の新しい証明書の生成
- シードクラスター上の **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た。
- シードクラスターに OADP Operator と Lifecycle Agent の最小バージョンをインストールした。
- シードクラスターに永続ボリュームが設定されていないことを確認する。
- Local Storage Operator が使用されている場合は、シードクラスターに **LocalVolume** CR が存在しないことを確認する。
- LVM ストレージが使用されている場合は、シードクラスターに **LVMCluster** CR が存在しないことを確認する。
- OADP が使用されている場合は、シードクラスターに **DataProtectionApplication** CR が存在しないことを確認する。

手順

1. クラスターをハブからデタッチして、シードイメージに含まれてはならない RHACM 固有のリソースをシードクラスターから削除します。
   a. 次のコマンドを実行してシードクラスターを手動でデタッチします。

      ```
      $ 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、シードクラスターの **SiteConfig** CR を **kustomization.yaml** から削除してクラスターをデタッチします。

      i. 複数の **SiteConfig** CR を参照する **kustomization.yaml** ファイルがある場合は、シードクラスターの **SiteConfig** CR を **kustomization.yaml** から削除します。

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

         generators:
         #- example-seed-sno1.yaml
         - example-target-sno2.yaml
         - example-target-sno3.yaml
         ```

      ii. 1 つの **SiteConfig** CR を参照する **kustomization.yaml** がある場合は、シードクラスターの **SiteConfig** CR を **kustomization.yaml** から削除し、**generators: {}** 行を追加します。

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

generators: {}
```

iii. Git リポジトリーで **kustomization.yaml** の変更をコミットし、変更をリポジトリーにプッシュします。
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. シードイメージをレジストリーにプッシュできるように、**Secret** オブジェクトを作成します。

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** リソースには、**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のレジストリーへの書き込みアクセス用 base64 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 **Secret** を適用します。

```
$ 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 の名前は **seedimage** 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を推奨します。

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している間、クラスターが再起動し、API 機能が失われます。**SeedGenerator** CR を適用すると、**kubelet** と CRI-O の操作が停止し、イメージ生成が開始されます。

より多くの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. クラスターが回復して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 stateroot 間の共有コンテナーパーティションの設定
- GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定

15.2.4. Lifecycle Agent を使用したイメージベースアップグレード用の ConfigMap オブジェクトの作成

Lifecycle Agent では、イメージベースアップグレード用に処理するために、すべての OADP リソース、追加のマニフェスト、およびカスタムカタログソースが **ConfigMap** オブジェクトにラップされている必要があります。

15.2.4.1. Lifecycle Agent を使用したイメージベース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ている。
- OADP バックアップおよび復元リソースを作成している。
- 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上に個別のパーティションを作成している。詳細は、「イメージベースアップグレード用の共有コンテナーパーティションの設定」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 Lifecycle Agent のバージョンをデプロイしている。
- 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ている。
- S3 互換の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ている。詳細は、「OADP のインストールについて」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 **Backup** 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 によって管理されている場合は、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 用の 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/k
usterlets.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、ハブクラスターの **open-cluster-management-agent** namespace にシークレットが存在しない場合は、**v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials** を削除します。

b. LVM Storage を介してクラスター上に永続ボリュームを作成した場合は、LVM Storage アーティファクトに次の YAML ファイルを追加します。

LVM Storage 用の 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** の値は、アプリケーションの **Restore** CR で指定された値よりも低くする必要があります。

2. 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. **openshift-adp** namespace に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用の OADP CR を作成します。

   LSO および LVM Storage のクラスタースコープ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.openshift.io/v
1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterroles/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 サンプルのリソース名を実際のリソースに置き換えます。

2 **lca.openshift.io/apply-wave** の値は、プラットフォームの **Restore** CR の値よりも大きく、アプリケーションの namespace スコープの **Restore** CR の値よりも小さくする必要があります。

b. namespace スコープのアプリケーションアーティファクト用の OADP CR を作成します。

LSO が使用される場合の OADP CR の 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 Storage 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
```

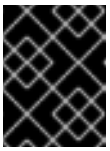
1 アプリケーションのカスタムリソースを定義します。

2 必須フィールド。

3 必須フィールド

4 LVM Storage 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になります。

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が、OpenShift Container Platform の現在の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 stateroot 間の共有コンテナーパーティションの設定
- 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アップグレード用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します。

手順

1. SR-IOV などの追加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 **ConfigMap** 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用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、アップグレード後もカスタムカタログソースを保持できます。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 **ConfigMap** 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 を使用したシングルノード OpenShift のイメージベースアップグレードの実行

15.2.5. GitOps ZTP を使用した Lifecycle Agent によるイメージベースアップグレード用の ConfigMap オブジェクトの作成

イメージベースアップグレードに備えて、**ConfigMap** オブジェクトにラップされた OADP リソース、追加のマニフェスト、カスタムカタログソースを作成します。

15.2.5.1. GitOps ZTP を使用したイメージベース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できるように 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。
- 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ている。
- 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上に個別のパーティションを作成している。詳細は、「GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定」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 Lifecycle Agent のバージョンをデプロイしている。
- 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ている。
- S3 互換の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ている。詳細は、「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。
- OADP **ConfigMap** を生成してクラスターにコピーできるように、OADP **ConfigMap** オブジェクト用の **openshift-adp** namespace が、すべてのマネージドクラスターとハブに存在する。

手順

1. ArgoCD ポリシーアプリケーションで使用する Git リポジトリーに次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
│   │   ├── PlatformBackupRestoreWithIBGU.yaml
├── ...
├── kustomization.yaml
```

source-crs/ibu/PlatformBackupRestoreWithIBGU.yaml ファイルは、ZTP コンテナーイメージで提供されます。

PlatformBackupRestoreWithIBGU.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-work:ibu-role,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、ハブクラスターの **open-cluster-management-agent** namespace にシークレットが存在しない場合は、**v1/secrets/open-cluster-management-agent/open-cluster-management-image-pull-credentials** を削除します。

注記

マネージドクラスターでイメージベースのアップグレードを直接実行する場合は、**PlatformBackupRestore.yaml** ファイルを使用します。

LVM Storage 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 Storage 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** の値は、アプリケーションの **Restore** CR で指定された値よりも低くする必要があります。

2. 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. **openshift-adp** namespace に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用の OADP CR を作成します。

   LSO および LVM Storage のクラスタースコープ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.openshift.io/v
1/securitycontextconstraints/test,rbac.authorization.k8s.io/v1/clusterroles/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) サンプルのリソース名を実際のリソースに置き換えます。

(2) **lca.openshift.io/apply-wave** の値は、プラットフォームの **Restore** CR の値よりも大きく、アプリケーションの namespace スコープの **Restore** CR の値よりも小さくする必要があります。

   b. namespace スコープのアプリケーションアーティファクト用の OADP CR を **source-crs/custom-crs** ディレクトリーに作成します。

   LSO が使用される場合の OADP CR の 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
```

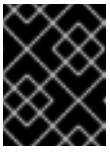
(1) アプリケーションのカスタムリソースを定義します。

LVM Storage 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 Storage 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になります。
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が、OpenShift Container Platform の現在の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. 次の内容を含む **kustomization.yaml** を作成します。

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

configMapGenerator: (1)
- files:
  - source-crs/ibu/PlatformBackupRestoreWithIBGU.yaml
```

```
    #- source-crs/custom-crs/ApplicationClusterScopedBackupRestore.yaml
    #- source-crs/custom-crs/ApplicationApplicationBackupRestoreLso.yaml
    name: oadp-cm
    namespace: openshift-adp 2
generatorOptions:
    disableNameSuffixHash: true
```

1 **Backup** および **Restore** CR を使用して、ハブクラスター上に **oadp-cm ConfigMap** オブジェクトを作成します。

2 OADP **ConfigMap** を生成してクラスターにコピーするために、この namespace がすべてのマネージドクラスターとハブに存在している必要があります。

4. 変更を Git リポジトリーにプッシュします。

関連情報

- GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定
- 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アップグレード用の追加マニフェストのラベル付け

追加のマニフェストにラベルを付けて、Lifecycle Agent が **lca.openshift.io/target-ocp-version: <target_version>** ラベルでラベル付けされたリソースを抽出できるようにします。

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。
- 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ている。
- 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上に個別のパーティションを作成している。詳細は、「GitOps ZTP を使用する場合の ostree stateroot 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。
- シードイメージで使用されるバージョンと互換性のある Lifecycle Agent のバージョンをデプロイしている。

手順

1. 既存のサイトの **PolicyGenTemplate** CR で、必要な追加マニフェストに **lca.openshift.io/target-ocp-version: <target_version>** 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** フィールドに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 のいずれかと一致す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 を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定
- GitOps ZTP を使用したシングルノード OpenShift クラスターのイメージベースアップグレードの実行

15.2.6. コンテナーストレージディスクの自動イメージクリーンアップの設定

使用可能なストレージ領域の最小しきい値をアノテーションによって設定することで、**Prep** ステージにおいて Lifecycle Agent が固定されていないイメージをクリーンアップするときの条件を設定します。デフォルトのコンテナーストレージディスク使用量のしきい値は 50% です。

Lifecycle Agent は、CRI-O に固定されているイメージや現在使用されているイメージを削除しません。Operator は、まずダングリング (dangling) 状態のイメージから削除するイメージを選択します。さらに、イメージの **Created** タイムスタンプに基づいて最も古いイメージから新しいイメージの順に並べ替えることで、削除するイメージを選択します。

15.2.6.1. コンテナーストレージディスクの自動イメージクリーンアップの設定

アノテーションにより、使用可能なストレージ領域の最小しきい値を設定します。

前提条件

- **ImageBasedUpgrade** CR を作成します。

手順

1. 次のコマンドを実行して、しきい値を 65% に増やします。

```
$ oc -n openshift-lifecycle-agent annotate ibu upgrade image-cleanup.lca.openshift.io/disk-usage-threshold-percent='65'
```

2. (オプション) 次のコマンドを実行して、しきい値のオーバーライドを削除します。

```
$ oc -n  openshift-lifecycle-agent annotate ibu upgrade image-cleanup.lca.openshift.io/disk-usage-threshold-percent-
```

15.2.6.2. コンテナーストレージディスクの自動イメージクリーンアップの無効化

自動イメージクリーンアップのしきい値を無効にします。

手順

1. 次のコマンドを実行して、自動イメージクリーンアップを無効にします。

```
$ oc -n openshift-lifecycle-agent annotate ibu upgrade image-cleanup.lca.openshift.io/on-prep='Disabled'
```

2. (オプション) 次のコマンドを実行して、自動イメージクリーンアップを再度有効にします。

```
$ oc -n  openshift-lifecycle-agent annotate ibu upgrade image-cleanup.lca.openshift.io/on-prep-
```

15.3. LIFECYCLE AGENT を使用したシングルノード OPENSHIFT クラスターのイメージベースアップグレードの実行

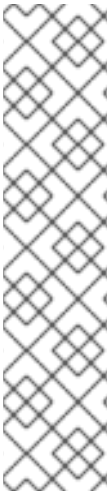
Lifecycle Agent を使用して、シングルノード OpenShift クラスターのイメージベースアップグレードを手動で実行できます。

クラスターに Lifecycle Agent をデプロイすると、**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、さまざまなステージを移動します。

15.3.1. Lifecycle Agent を使用したイメージベースアップグレードの Prep ステージへの移行

クラスターに Lifecycle Agent をデプロイすると、**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ら、**Prep** ステージに進むことができます。詳細は、「Lifecycle Agent を使用したイメージベース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。



注記

非接続環境で、シードクラスターのリリースイメージレジストリーがターゲットクラスターのリリースイメージレジストリーと異なる場合は、**ImageDigestMirrorSet** (IDMS) リソースを作成して、別のミラーリングされるリポジトリーの場所を設定する必要があります。詳細は、「イメージレジストリーのリポジトリーミラーリングの設定」を参照してください。

次のコマンドを実行すると、シードイメージで使用されているリリースレジストリーを取得できます。

```
$ skopeo inspect docker://<imagename> | jq -r '.Labels."com.openshift.lifecycle-agent.seed_cluster_info" | fromjson | .release_registry'
```

前提条件

- クラスターをバックアップおよび復元するためのリソースを作成している。

手順

1. **ImageBasedUpgrade** CR にパッチが適用され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。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内にある場合は、コンテナーイメージをプルするための認証情報を含む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、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含む **ConfigMap** リソースのリストを指定します。

(6) OADP **ConfigMap** 情報を含む **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 リソースと追加のマニフェストの **ConfigMap** オブジェクトを指定すると、Lifecycle Agent は **Prep** ステージで指定された **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 Agent が **extraManifests** パラメーターに問題を検出した場合の検証警告またはエラー。
- Lifecycle Agent が **oadpContent** パラメーターに問題を検出した場合の検証エラー。

検証警告は **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。これらの警告 (CRD の欠落、namespace の欠落、またはドライランの失敗など) は、警告に関する詳細で **Prep** ステージの **status.conditions** と **ImageBasedUpgrade** CR の **annotation** フィールドを更新します。

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の検証エラーが発生すると、**Prep** ステージが失敗し、**Upgrade** ステージがブロックされます。

検証に合格すると、クラスターは新しい **ostree** stateroot を作成します。これには、シードイメージのプルと展開、およびホストレベルのコマンドの実行が含まれます。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アップグレード用の ConfigMap オブジェクトの作成
- イメージレジストリーのリポジトリーミラーリングの設定

15.3.2. Lifecycle Agent を使用したイメージベースアップグレードの Upgrade ステージへの移行

シードイメージを生成し、**Prep** ステージを完了したら、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、その後、Lifecycle Agent 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または停止した場合は、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、手動で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、「Lifecycle Agent を使用したイメージベースアップグレードの Rollback ステージへの移行」を参照してください。

前提条件

- **Prep** ステージを完了している。

手順

1. **Upgrade** ステージに移動するには、次のコマンドを実行して、**ImageBasedUpgrade** CR の **stage** フィールドの値を **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** および **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が再起動します。

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. 正常にアップグレードされたら、次のコマンドを実行して、**ImageBasedUpgrade** CR の **stage** フィールドの値を **Idle** にパッチして、変更を終了します。

```
$ 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** ステージに移行すると、変更をロールバックすることはできません。

Lifecycle Agent は、アップグレードプロセス中に作成されたすべてのリソースを削除します。

5. アップグレードが成功したら、OADP Operator とその設定ファイルを削除できます。詳細は、「クラスターからの Operator の削除」を参照してください。

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、クラスターの復元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 環境にのみ固有です。

関連情報

- Lifecycle Agent を使用したイメージベースアップグレードの Rollback ステージへの移行
- クラスターからの Operator の削除

15.3.3. Lifecycle Agent を使用したイメージベースアップグレードの Rollback ステージへの移行

再起動後、**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は、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、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できない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ハブクラスターにログインしている。
- 元の stateroot 上のコントロールプレーン証明書が有効である。証明書の有効期限が切れている場合は、「コントロールプレーン証明書の期限切れの状態からのリカバリー」を参照してください。

手順

1. Rollback ステージに移行するには、次のコマンドを実行して、**ImageBasedUpgrade** CR の **stage** フィールドの値を **Rollback** にパッチします。

```
$ 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 を使用して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. 正常に変更されたら、次のコマンドを実行して、**ImageBasedUpgrade** CR の **stage** フィールドの値を **Idle** にパッチして、ロールバックを終了します。

```
$ 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** ステージに移行すると、Lifecycle Agent 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。

関連情報

- コントロールプレーン証明書の期限切れの状態からのリカバリー

15.3.4. Lifecycle Agent を使用したイメージベースアップグレードのトラブルシューティング

問題の影響を受けるマネージドクラスターでトラブルシューティング手順を実行します。

重要

ImageBasedGroupUpgrade CR を使用してクラスターをアップグレードする場合は、マネージドクラスターでトラブルシューティングまたは復元手順を実行した後、**lcm.openshift.io/ibgu-<stage>-completed** または **lcm.openshift.io/ibgu-<stage>-failed** クラスターラベルが適切に更新されていることを確認してください。これにより、TALM がクラスターのイメージベースのアップグレードを引き続き管理できるようになります。

15.3.4.1. ログの収集

oc adm must-gather CLI を使用して、デバッグとトラブルシューティングのため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。

15.3.4.2. AbortFailed または FinalizeFailed エラー

問題

最終ステージの間、または **Prep** ステージでプロセスを停止すると、Lifecycle Agent 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 が上記の手順を実行できない場合は、**AbortFailed** または **FinalizeFailed** 状態に移行します。条件メッセージとログには、どの手順が失敗したかが表示されます。

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、失敗が発生した理由を特定します。
2. Lifecycle Agent にクリーンアップを再試行するように指示するには、**ImageBasedUpgrade** CR に **lca.openshift.io/manual-cleanup-done** アノテーションを追加します。
   このアノテーションを確認した後、Lifecycle Agent はクリーンアップを再試行し、成功した場合は **ImageBasedUpgrade** ステージが **Idle** に移行します。

   クリーンアップが再度失敗した場合は、リソースを手動でクリーンアップできます。

15.3.4.2.1. 手動での stateroot のクリーンアップ

問題

Lifecycle Agent は **Prep** ステージで停止し、新しい stateroot をクリーンアップします。アップグレードまたはロールバックが成功した後に終了すると、Lifecycle Agent は古い stateroot をクリーンアップします。この手順が失敗した場合は、ログを調べて失敗の原因を特定することを推奨します。

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確認します。

```
$ ostree admin status
```

2. ある場合は、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、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

起動され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 リソースを手動でクリーンアップする

問題

Lifecycle Agent と S3 バックエンド間の接続の問題により、OADP リソースの自動クリーンアップが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ことで、Lifecycle Agent 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 Storage ボリュームの内容が復元されない

LVM Storage 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、LVM Storage が正しく設定されていないと、永続ボリュームの内容が復元されない可能性があります。

15.3.4.3.1. Backup CR に LVM Storage 関連のフィールドがない

問題

Backup CR に、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。以下を実行すると、アプリケーション Pod 内のイベントをチェックして、この問題が発生しているか確認できます。

```
$ oc describe pod <your_app_name>
```

Backup CR に LVM Storage 関連のフィールドがないことを示す出力例

```
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 **Backup** CR に **logicalvolumes.topolvm.io** を含める必要があります。このリソースがない場合、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しく復元しますが、この永続ボリュームに関連付けられた **logicalvolume** 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、このセクションを次のように設定する必要があります。

15.3.4.3.2. Restore CR に LVM Storage 関連のフィールドがない

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、アップグレード後に永続ボリュームの内容は保持されません。

1. ピボット前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リスト表示します。

```
$ 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リスト表示します。

```
$ 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

この問題は、**logicalvolume** ステータスが **Restore** CR に保存されないことが原因となっています。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 には、次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、このセクションを次のように設定する必要があります。

15.3.4.4. 失敗した Backup CR および Restore CR のデバッグ

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** 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 を説明します。

```
$ 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 を説明します。

```
$ 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 を使用したシングルノード OPENSHIFT クラスターのイメージベースアップグレードの実行

ハブクラスター上の1つのリソース、**ImageBasedGroupUpgrade** カスタムリソース (CR) を使用して、すべてのステージを通じて、選択したマネージドクラスターのグループでイメージベースのアップグレードを管理できます。Topology Aware Lifecycle Manager (TALM) は、**ImageBasedGroupUpgrade** CR を調整し、手動制御のアップグレードフローまたは完全に自動のアップグレードフローにより、定められたステージ遷移を完了するための基盤リソースを作成します。

イメージベースのアップグレードの詳細は、「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」を参照してください。

関連情報

- 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

15.4.1. ハブ上の ImageBasedGroupUpgrade CR を使用してイメージベースのアップグレードを大規模に管理する

ImageBasedGroupUpgrade CR は、**ImageBasedUpgrade** API と **ClusterGroupUpgrade** API を組み合わせたものです。たとえば、**ImageBasedGroupUpgrade** API でも、**ClusterGroupUpgrade** API と同じ方法で、クラスターの選択とロールアウトストラテジーを定義できます。ステージ遷移は **ImageBasedUpgrade** API とは異なります。ImageBasedGroupUpgrade API を使用すると、アクションとも呼ばれる複数のステージ遷移を、1つのロールアウトストラテジーを共有する1つのステップに組み合わせることができます。

ImageBasedGroupUpgrade.yaml の例

```
apiVersion: lcm.openshift.io/v1alpha1
kind: ImageBasedGroupUpgrade
metadata:
  name: <filename>
  namespace: default
spec:
  clusterLabelSelectors: (1)
    - matchExpressions:
      - key: name
        operator: In
        values:
        - spoke1
        - spoke4
        - spoke6
  ibuSpec:
    seedImageRef: (2)
      image: quay.io/seed/image:4.17.0-rc.1
      version: 4.17.0-rc.1
      pullSecretRef:
        name: "<seed_pull_secret>"
    extraManifests: (3)
      - name: example-extra-manifests
        namespace: openshift-lifecycle-agent
    oadpContent: (4)
      - name: oadp-cm
        namespace: openshift-adp
  plan: (5)
    - actions: ["Prep", "Upgrade", "FinalizeUpgrade"]
```

```
  rolloutStrategy:
    maxConcurrency: 200 6
    timeout: 2400 7
```

1 アップグレードするクラスター。

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、およびイメージにアクセスするために必要なシークレット。

3 オプション: シードイメージに含まれていない追加のマニフェストをターゲットクラスターに適用します。カスタムカタログソースの **ConfigMap** オブジェクトも適用します。

4 OADP の **Backup** および **Restore** CR を含む **ConfigMap** リソース。

5 アップグレードプランの詳細。

6 バッチで更新するクラスターの数。

7 アクションを完了するまでのタイムアウト制限 (分単位)。

15.4.1.1. サポートされているアクションの組み合わせ

アクションは、選択されたクラスターグループのアップグレードプランの各ステップで TALM が実行する一連のステージ遷移です。**ImageBasedGroupUpgrade** CR 内の **action** エントリーは、それぞれ個別のステップです。1 つのステップに、同じロールアウトストラテジーを共有する 1 つ以上のアクションを含めます。アクションをステップに分けることで、各アクションのロールアウトストラテジーをより細かく制御できます。

これらのアクションは、アップグレードプラン内でさまざまな方法で組み合わせることができます。後続のステップは、後で追加することができます。プランにステップを追加する前に、前のステップが完了するか失敗するまで待機します。前のステップで失敗したクラスターに対して追加するステップの最初のアクションは、**Abort** または **Rollback** のどちらかである必要があります。

重要

進行中のプランからアクションまたはステップを削除することはできません。

次の表にプランの例を示します。それぞれの例で、ロールアウトストラテジーに対する制御レベルが異なります。

表15.5 アップグレードプランの例

プランの例	説明
plan: - actions: ["Prep", "Upgrade", "FinalizeUpgrade"] rolloutStrategy: maxConcurrency: 200 timeout: 60	すべてのアクションが同じストラテジーを共有しています。

プランの例	説明

plan:
- actions: ["Prep", "Upgrade"]
  rolloutStrategy:
  maxConcurrency: 200
  timeout: 60
- actions: ["FinalizeUpgrade"]
  rolloutStrategy:
  maxConcurrency: 500
  timeout: 10
	いくつかのアクションが同じストラテジーを共有しています。

plan:
- actions: ["Prep"]
  rolloutStrategy:
  maxConcurrency: 200
  timeout: 60
- actions: ["Upgrade"]
  rolloutStrategy:
  maxConcurrency: 200
  timeout: 20
- actions: ["FinalizeUpgrade"]
  rolloutStrategy:
  maxConcurrency: 500
  timeout: 10
	すべてのアクションに異なるストラテジーがあります。

重要

クラスターは、いずれかのアクションに失敗すると、同じステップの残りのアクションをスキップします。

ImageBasedGroupUpgrade API は次のアクションを受け入れます。

Prep

Prep ステージに進み、アップグレードリソースの準備を開始します。

Upgrade

Upgrade ステージに移行して、アップグレードを開始します。

FinalizeUpgrade

Idle ステージに移行して、**Upgrade** アクションを完了した特定のクラスターのアップグレードを確定します。

Rollback

Rollback ステージに移行して、正常にアップグレードされたクラスターでのみロールバックを開始します。

FinalizeRollback

Idle ステージに移行して、ロールバックを完了します。

AbortOnFailure

Idle ステージに移行して、**Prep** または **Upgrade** アクションに失敗した特定のクラスターのアップグレードをキャンセルします。

Abort

Idle ステージに移行して、まだアップグレードされていないクラスターでのみ、進行中のアップグレードをキャンセルします。

次のアクションの組み合わせがサポートされています。角括弧のペアは、**plan** セクションの 1 つのステップを表します。

- **["Prep"]、["Abort"]**
- **["Prep", "Upgrade", "FinalizeUpgrade"]**
- **["Prep"]、["AbortOnFailure"]、["Upgrade"]、["AbortOnFailure"]、["FinalizeUpgrade"]**
- **["Rollback", "FinalizeRollback"]**

完全に新しい **ImageBasedGroupUpgrade** CR から進行中のアップグレードを再開またはキャンセルする必要がある場合は、次のいずれかの組み合わせを使用します。

- **["Upgrade","FinalizeUpgrade"]**
- **["FinalizeUpgrade"]**
- **["FinalizeRollback"]**
- **["Abort"]**
- **["AbortOnFailure"]**

15.4.1.2. クラスター選択のためのラベル付け

最初のクラスターの選択には、**spec.clusterLabelSelectors** フィールドを使用します。さらに、TALM が、最後のステージ遷移の結果に応じて、マネージドクラスターにラベルを付けます。

ステージが完了または失敗すると、関連するクラスターに TALM が次のラベルを付けます。

- **lcm.openshift.io/ibgu-<stage>-completed**
- **lcm.openshift.io/ibgu-<stage>-failed**

これらのクラスターラベルは、発生した問題をトラブルシューティングした後、クラスターのグループのアップグレードをキャンセルまたはロールバックするために使用します。

重要

ImageBasedGroupUpgrade CR を使用してクラスターをアップグレードする場合は、マネージドクラスターでトラブルシューティングまたは復元手順を実行した後、**lcm.openshift.io/ibgu-<stage>-completed** または **lcm.openshift.io/ibgu-<stage>-failed** クラスターラベルが適切に更新されていることを確認してください。これにより、TALM がクラスターのイメージベースのアップグレードを引き続き管理できるようになります。

たとえば、アップグレードが正常に完了したクラスターを除くすべてのマネージドクラスターのアップグレードをキャンセルする場合は、プランに **Abort** アクションを追加できます。**Abort** アクションは、**ImageBasedUpgrade** CR を **Idle** ステージに戻し、まだアップグレードされていないクラスターのアップグレードをキャンセルします。個別の **Abort** アクションを追加すると、TALM が **lcm.openshift.io/ibgu-upgrade-completed** ラベルを持つクラスターに対して **Abort** アクションを実行しなくなります。

アップグレードが正常にキャンセルされるか完了すると、クラスターラベルが削除されます。

15.4.1.3. ステータスの監視

ImageBasedGroupUpgrade CR は、すべてのクラスターの包括的なステータスレポートを 1 カ所に集約し、より優れた監視エクスペリエンスを実現します。監視できるアクションは次のとおりです。

status.clusters.completedActions

plan セクションで定義されている完了したすべてのアクションを表示します。

status.clusters.currentAction

現在進行中のすべてのアクションを表示します。

status.clusters.failedActions

詳細なエラーメッセージとともに、失敗したすべてのアクションを表示します。

15.4.2. 複数のステップでマネージドクラスターのイメージベースのアップグレードを大規模に実行する

アップグレードによってサービスが中断されるタイミングを適切に制御する必要があるユースケースの場合は、**ImageBasedGroupUpgrade** CR を使用し、前のステップが完了した後にアクションを追加することで、マネージドクラスターのセットをアップグレードできます。前のステップの結果を評価した後、次のアップグレードステージに進むことも、手順全体の失敗したステップをトラブルシューティングすることもできます。

重要

特定のアクションの組み合わせのみがサポートされています。これは **サポートされているアクションの組み合わせ** に記載されています。

前提条件

- **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。
- イメージベースのアップグレードで使用するリソースのポリシーと **ConfigMap** オブジェクトを作成した。
- ハブクラスターを介して、すべてのマネージドクラスターに Lifecycle Agent と OADP Operator をインストールした。

手順

1. ハブクラスターで、**ImageBasedGroupUpgrade** CR を含む YAML ファイルを作成します。

```
apiVersion: lcm.openshift.io/v1alpha1
kind: ImageBasedGroupUpgrade
metadata:
```

```
  name: <filename>
  namespace: default
spec:
  clusterLabelSelectors: 1
    - matchExpressions:
      - key: name
        operator: In
        values:
        - spoke1
        - spoke4
        - spoke6
  ibuSpec:
    seedImageRef: 2
      image: quay.io/seed/image:4.16.0-rc.1
      version: 4.16.0-rc.1
      pullSecretRef:
        name: "<seed_pull_secret>"
    extraManifests: 3
      - name: example-extra-manifests
        namespace: openshift-lifecycle-agent
    oadpContent: 4
      - name: oadp-cm
        namespace: openshift-adp
  plan: 5
    - actions: ["Prep"]
      rolloutStrategy:
        maxConcurrency: 2
        timeout: 2400
```

1. アップグレードするクラスター。
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、およびイメージにアクセスするために必要なシークレット。
3. オプション: シードイメージに含まれていない追加のマニフェストをターゲットクラスターに適用します。カスタムカタログソースの **ConfigMap** オブジェクトも適用します。
4. OADP の **Backup** および **Restore** CR を含む **ConfigMap** リソース。
5. アップグレードプランの詳細。

2. ハブクラスターで次のコマンドを実行して、作成したファイルを適用します。

```
$ oc apply -f <filename>.yaml
```

3. ハブクラスターで次のコマンドを実行して、ステータスの更新を監視します。

```
$ oc get ibgu -o yaml
```

出力例

```
# ...
status:
  clusters:
```

```
  - completedActions:
    - action: Prep
    name: spoke1
  - completedActions:
    - action: Prep
    name: spoke4
  - failedActions:
    - action: Prep
    name: spoke6
# ...
```

上記のサンプルプランの出力は、始めは **Prep** ステージだけです。前のステップの結果に基づいて、このプランにアクションを追加してください。TALM により、アップグレードが成功したか失敗したかを示すラベルがクラスターに追加されます。たとえば、**Prep** ステージに失敗したクラスターには、**lcm.openshift.io/ibgu-prep-failed** が適用されます。

失敗を調査した後、アップグレードプランに **AbortOnFailure** ステップを追加できます。このステップにより、**lcm.openshift.io/ibgu-<action>-failed** というラベルの付いたクラスターが **Idle** ステージに戻されます。選択されたクラスターのアップグレードに関連するリソースがすべて削除されます。

4. オプション: 次のコマンドを実行して、既存の **ImageBasedGroupUpgrade** CR に **AbortOnFailure** アクションを追加します。

```
$ oc patch ibgu <filename> --type=json -p \
'[{"op": "add", "path": "/spec/plan/-", "value": {"actions": ["AbortOnFailure"], "rolloutStrategy":
{"maxConcurrency": 5, "timeout": 10}}}]'
```

   a. 次のコマンドを実行して、ステータスの更新の監視を継続します。

```
$ oc get ibgu -o yaml
```

5. 次のコマンドを実行して、既存の **ImageBasedGroupUpgrade** CR にアクションを追加します。

```
$ oc patch ibgu <filename> --type=json -p \
'[{"op": "add", "path": "/spec/plan/-", "value": {"actions": ["Upgrade"], "rolloutStrategy":
{"maxConcurrency": 2, "timeout": 30}}}]'
```

6. オプション: 次のコマンドを実行して、既存の **ImageBasedGroupUpgrade** CR に **AbortOnFailure** アクションを追加します。

```
$ oc patch ibgu <filename> --type=json -p \
'[{"op": "add", "path": "/spec/plan/-", "value": {"actions": ["AbortOnFailure"], "rolloutStrategy":
{"maxConcurrency": 5, "timeout": 10}}}]'
```

   a. 次のコマンドを実行して、ステータスの更新の監視を継続します。

```
$ oc get ibgu -o yaml
```

7. 次のコマンドを実行して、既存の **ImageBasedGroupUpgrade** CR にアクションを追加します。

```
$ oc patch ibgu <filename> --type=json -p \
'[{"op": "add", "path": "/spec/plan/-", "value": {"actions": ["FinalizeUpgrade"], "rolloutStrategy":
{"maxConcurrency": 10, "timeout": 3}}}]'
```

検証

- 次のコマンドを実行してステータスの更新を監視します。

```
$ oc get ibgu -o yaml
```

出力例

```
# ...
status:
  clusters:
  - completedActions:
    - action: Prep
    - action: AbortOnFailure
    failedActions:
    - action: Upgrade
    name: spoke1
  - completedActions:
    - action: Prep
    - action: Upgrade
    - action: FinalizeUpgrade
    name: spoke4
  - completedActions:
    - action: AbortOnFailure
    failedActions:
    - action: Prep
    name: spoke6
# ...
```

関連情報

- GitOps ZTP を使用した場合の ostree stateroot 間の共有コンテナーパーティションの設定
- GitOps ZTP を使用した Lifecycle Agent によるイメージベースアップグレード用の ConfigMap オブジェクトの作成
- 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
- Backup CR の作成
- Restore CR の作成
- サポートされているアクションの組み合わせ

15.4.3. 1 つのステップでマネージドクラスターのイメージベースのアップグレードを大規模に実行する

サービスの中断が問題にならないユースケースの場合は、**ImageBasedGroupUpgrade** CR を使用し、1 つのロールアウトストラテジーによって複数のアクションを 1 つのステップに組み合わせることで、マネージドクラスターのセットをアップグレードできます。ロールアウトストラテジーを 1 つにすると、

アップグレード時間を短縮できます。ただし、アップグレードプランが完了した後でないと、失敗したクラスターのトラブルシューティングを行うことができません。

前提条件

- **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。
- イメージベースのアップグレードで使用するリソースのポリシーと **ConfigMap** オブジェクトを作成した。
- ハブクラスターを介して、すべてのマネージドクラスターに Lifecycle Agent と OADP Operator をインストールした。

手順

1. ハブクラスターで、**ImageBasedGroupUpgrade** CR を含む YAML ファイルを作成します。

```
apiVersion: lcm.openshift.io/v1alpha1
kind: ImageBasedGroupUpgrade
metadata:
  name: <filename>
  namespace: default
spec:
  clusterLabelSelectors: 1
    - matchExpressions:
      - key: name
        operator: In
        values:
        - spoke1
        - spoke4
        - spoke6
  ibuSpec:
    seedImageRef: 2
      image: quay.io/seed/image:4.17.0-rc.1
      version: 4.17.0-rc.1
      pullSecretRef:
        name: "<seed_pull_secret>"
    extraManifests: 3
      - name: example-extra-manifests
        namespace: openshift-lifecycle-agent
    oadpContent: 4
      - name: oadp-cm
        namespace: openshift-adp
  plan: 5
    - actions: ["Prep", "Upgrade", "FinalizeUpgrade"]
      rolloutStrategy:
        maxConcurrency: 200 6
        timeout: 2400 7
```

1. アップグレードするクラスター。
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、およびイメージにアクセスするために必要なシークレット。

3 オプション: シードイメージに含まれていない追加のマニフェストをターゲットクラスターに適用します。カスタムカタログソースの **ConfigMap** オブジェクトも適用します。

4 OADP の **Backup** および **Restore** CR を含む **ConfigMap** リソース。

5 アップグレードプランの詳細。

6 バッチで更新するクラスターの数。

7 アクションを完了するまでのタイムアウト制限 (分単位)。

2. ハブクラスターで次のコマンドを実行して、作成したファイルを適用します。

```
$ oc apply -f <filename>.yaml
```

検証

- 次のコマンドを実行してステータスの更新を監視します。

```
$ oc get ibgu -o yaml
```

出力例

```
# ...
status:
  clusters:
  - completedActions:
    - action: Prep
    failedActions:
    - action: Upgrade
    name: spoke1
  - completedActions:
    - action: Prep
    - action: Upgrade
    - action: FinalizeUpgrade
    name: spoke4
  - failedActions:
    - action: Prep
    name: spoke6
# ...
```

15.4.4. マネージドクラスターのイメージベースのアップグレードを大規模にキャンセルする

Prep ステージを完了したマネージドクラスターのセットのアップグレードをキャンセルできます。

重要

特定のアクションの組み合わせのみがサポートされています。これは **サポートされているアクションの組み合わせ** に記載されています。

前提条件

- **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. ハブクラスターで、**ImageBasedGroupUpgrade** CR を含む別の YAML ファイルを作成します。

```
apiVersion: lcm.openshift.io/v1alpha1
kind: ImageBasedGroupUpgrade
metadata:
  name: <filename>
  namespace: default
spec:
  clusterLabelSelectors:
    - matchExpressions:
      - key: name
        operator: In
        values:
        - spoke4
  ibuSpec:
    seedImageRef:
      image: quay.io/seed/image:4.16.0-rc.1
      version: 4.16.0-rc.1
      pullSecretRef:
        name: "<seed_pull_secret>"
    extraManifests:
      - name: example-extra-manifests
        namespace: openshift-lifecycle-agent
    oadpContent:
      - name: oadp-cm
        namespace: openshift-adp
  plan:
    - actions: ["Abort"]
      rolloutStrategy:
        maxConcurrency: 5
        timeout: 10
```

Prep ステージを完了したすべてのマネージドクラスターが、**Idle** ステージに戻ります。

2. ハブクラスターで次のコマンドを実行して、作成したファイルを適用します。

```
$ oc apply -f <filename>.yaml
```

検証

- 次のコマンドを実行してステータスの更新を監視します。

```
$ oc get ibgu -o yaml
```

出力例

```
# ...
status:
  clusters:
```

```
  - completedActions:
    - action: Prep
    currentActions:
    - action: Abort
    name: spoke4
# ...
```

関連情報

- サポートされているアクションの組み合わせ

15.4.5. マネージドクラスターのイメージベースのアップグレードを大規模にロールバックする

アップグレードが成功した後に解決できない問題が発生した場合は、管理対象クラスターのセットの変更をロールバックします。別の **ImageBasedGroupUpgrade** CR を作成し、ロールバックするマネージドクラスターのセットを定義する必要があります。

重要

特定のアクションの組み合わせのみがサポートされています。これは **サポートされているアクションの組み合わせ** に記載されています。

前提条件

- **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. ハブクラスターで、**ImageBasedGroupUpgrade** CR を含む別の YAML ファイルを作成します。

```
apiVersion: lcm.openshift.io/v1alpha1
kind: ImageBasedGroupUpgrade
metadata:
  name: <filename>
  namespace: default
spec:
  clusterLabelSelectors:
    - matchExpressions:
      - key: name
        operator: In
        values:
        - spoke4
  ibuSpec:
    seedImageRef:
      image: quay.io/seed/image:4.17.0-rc.1
      version: 4.17.0-rc.1
      pullSecretRef:
        name: "<seed_pull_secret>"
    extraManifests:
      - name: example-extra-manifests
        namespace: openshift-lifecycle-agent
    oadpContent:
```

```
      - name: oadp-cm
        namespace: openshift-adp
  plan:
    - actions: ["Rollback", "FinalizeRollback"]
      rolloutStrategy:
        maxConcurrency: 200
        timeout: 2400
```

2. ハブクラスターで次のコマンドを実行して、作成したファイルを適用します。

```
$ oc apply -f <filename>.yaml
```

定義したラベルに一致するすべてのマネージドクラスターが、**Rollback** ステージに戻されます。その後 **Idle** ステージに移行してロールバックが完了します。

検証

- 次のコマンドを実行してステータスの更新を監視します。

```
$ oc get ibgu -o yaml
```

出力例

```
# ...
status:
  clusters:
  - completedActions:
    - action: Rollback
    - action: FinalizeRollback
    name: spoke4
# ...
```

関連情報

- サポートされているアクションの組み合わせ
- コントロールプレーン証明書の期限切れの状態からのリカバリー

15.4.6. Lifecycle Agent を使用したイメージベースアップグレードのトラブルシューティング

問題の影響を受けるマネージドクラスターでトラブルシューティング手順を実行します。

重要

ImageBasedGroupUpgrade CR を使用してクラスターをアップグレードする場合は、マネージドクラスターでトラブルシューティングまたは復元手順を実行した後、**lcm.openshift.io/ibgu-<stage>-completed** または **lcm.openshift.io/ibgu-<stage>-failed** クラスターラベルが適切に更新されていることを確認してください。これにより、TALM がクラスターのイメージベースのアップグレードを引き続き管理できるようになります。

15.4.6.1. ログの収集

oc adm must-gather CLI を使用して、デバッグとトラブルシューティングのため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。

15.4.6.2. AbortFailed または FinalizeFailed エラー

問題

最終ステージの間、または **Prep** ステージでプロセスを停止すると、Lifecycle Agent 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 が上記の手順を実行できない場合は、**AbortFailed** または **FinalizeFailed** 状態に移行します。条件メッセージとログには、どの手順が失敗したかが表示されます。

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、失敗が発生した理由を特定します。

2. Lifecycle Agent にクリーンアップを再試行するように指示するには、**ImageBasedUpgrade** CR に **lca.openshift.io/manual-cleanup-done** アノテーションを追加します。
このアノテーションを確認した後、Lifecycle Agent はクリーンアップを再試行し、成功した場合は **ImageBasedUpgrade** ステージが **Idle** に移行します。

クリーンアップが再度失敗した場合は、リソースを手動でクリーンアップできます。

15.4.6.2.1. 手動での stateroot のクリーンアップ

問題

Lifecycle Agent は **Prep** ステージで停止し、新しい stateroot をクリーンアップします。アップグレードまたはロールバックが成功した後に終了すると、Lifecycle Agent は古い stateroot をクリーンアップします。この手順が失敗した場合は、ログを調べて失敗の原因を特定することを推奨します。

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確認します。

```
$ ostree admin status
```

2. ある場合は、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、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**
>
> 起動され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}"
```

15.4.6.2.2. OADP リソースを手動でクリーンアップする

問題

Lifecycle Agent と S3 バックエンド間の接続の問題により、OADP リソースの自動クリーンアップが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ことで、Lifecycle Agent 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。

15.4.6.3. LVM Storage ボリュームの内容が復元されない

LVM Storage 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、LVM Storage が正しく設定されていないと、永続ボリュームの内容が復元されない可能性があります。

15.4.6.3.1. Backup CR に LVM Storage 関連のフィールドがない

問題

Backup CR に、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。以下を実行すると、アプリケーション Pod 内のイベントをチェックして、この問題が発生しているか確認できます。

```
$ oc describe pod <your_app_name>
```

Backup CR に LVM Storage 関連のフィールドがないことを示す出力例

```
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 **Backup** CR に **logicalvolumes.topolvm.io** を含める必要があります。このリソースがない場合、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しく復元しますが、この永続ボリュームに関連付けられた **logicalvolume** 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、このセクションを次のように設定する必要があります。

15.4.6.3.2. Restore CR に LVM Storage 関連のフィールドがない

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、アップグレード後に永続ボリュームの内容は保持されません。

1. ピボット前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リスト表示します。

```
$ 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リスト表示します。

```
$ 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

この問題は、**logicalvolume** ステータスが **Restore** CR に保存されないことが原因となっています。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 には、次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、このセクションを次のように設定する必要があります。

15.4.6.4. 失敗した Backup CR および Restore CR のデバッグ

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** 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 を説明します。

```
$ 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 を説明します。

```
$ 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
```

第16章 通信事業者コア CNF クラスターの DAY 2 オペレーション

16.1. 通信事業者向けコア CNF クラスターのアップグレード

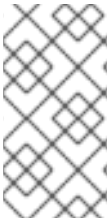
16.1.1. 通信事業者向けコア CNF クラスターのアップグレード

OpenShift Container Platform には、すべての偶数番号のリリースと EUS リリース間の更新パスに対する長期サポート (Extended Update Support (EUS)) があります。ある EUS バージョンから次の EUS バージョンに更新できます。y-stream バージョンと z-stream バージョン間で更新することも可能です。

16.1.1.1. 通信事業者コア CNF クラスターのクラスター更新

クラスターの更新は、バグや潜在的なセキュリティーの脆弱性を確実に修正するうえで重要なタスクです。多くの場合、クラウドネイティブネットワーク機能 (CNF) の更新には、クラスターバージョンの更新時に提供されるプラットフォームの追加機能が必要になります。また、クラスタープラットフォームがサポート対象のバージョンになるように、クラスターを定期的に更新する必要があります。

EUS リリースを最新の状態に保ち、一部の重要な z-stream リリースにのみアップグレードすることで、更新により最新の状態に保つのに必要な労力を最小限に抑えることができます。



注記

クラスターの更新パスは、クラスターのサイズとトポロジーによって異なります。ここで説明する更新手順は、3 ノードクラスターから、通信事業者スケールチームによって認定された最大規模のクラスターまで、ほとんどのクラスターに適用されます。これには、混合ワークロードクラスターのシナリオがいくつか含まれています。

説明する更新シナリオは次のとおりです。

- コントロールプレーンのみの更新
- y-stream の更新
- z-stream の更新

重要

コントロールプレーンのみの更新は、以前は EUS から EUS への更新と呼ばれていました。コントロールプレーンのみの更新は、OpenShift Container Platform の偶数番号のマイナーバージョン間でのみ実行可能です。

16.1.2. 更新バージョン間のクラスター API バージョンの検証

API は、コンポーネントの更新に伴い、次第に変更されます。クラウドネイティブネットワーク機能 (CNF) API と更新されたクラスターバージョンの間に互換性があることを検証することが重要です。

16.1.2.1. OpenShift Container Platform API の互換性

新しい y-stream 更新の一部としてどの z-stream リリースに更新するかを検討するときは、移行元の z-stream バージョンにあるすべてのパッチが新しい z-stream バージョンにあることを確認する必要があります。更新後のバージョンに必要なパッチがすべて含まれていない場合、Kubernetes の組み込みの互換性が損なわれます。

たとえば、クラスターのバージョンが 4.15.32 の場合、4.15.32 に適用されているすべてのパッチを含む 4.16 z-stream リリースに更新する必要があります。

16.1.2.1.1. Kubernetes のバージョンスキューについて

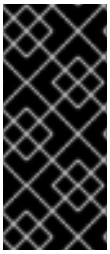
クラスター Operator は、それぞれ特定の API バージョンをサポートしています。Kubernetes API は時間とともに進化し、新しいバージョンが非推奨になったり、既存の API が変更されたりする可能性があります。これは "バージョンスキュー" と呼ばれます。API の変更は、新しいリリースごとに確認する必要があります。API には Operator の複数のリリース間で互換性がある場合もありますが、互換性は保証されていません。バージョンスキューから生じる問題を軽減するために、明確な更新ストラテジーに従ってください。

関連情報

- API 階層について
- Kubernetes バージョンスキューポリシー

16.1.2.2. クラスターのバージョン更新パスの確認

Red Hat OpenShift Container Platform Update Graph ツールを使用して、更新先の z-stream リリースに対してパスが有効かどうかを確認します。Red Hat テクニカルアカウントマネージャーに更新の確認を取り、更新パスが通信事業者の実装で有効であることを確認してください。

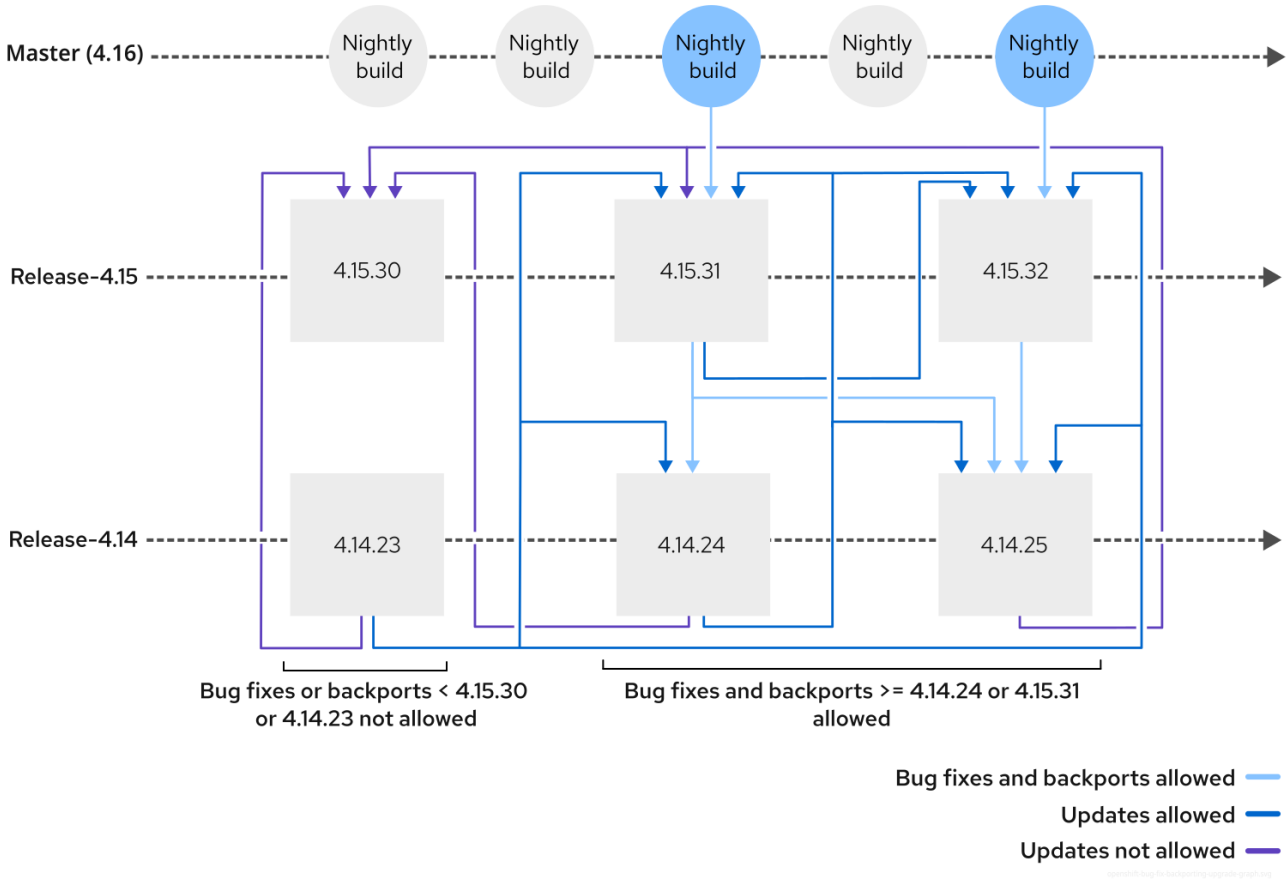


重要

更新先の <4.y+1.z> または <4.y+2.z> バージョンは、更新元の <4.y.z> リリースと同じパッチレベルである必要があります。

OpenShift の更新プロセスでは、特定の <4.y.z> リリースに修正が存在する場合、更新先の <4.y+1.z> リリースにもその修正が存在する必要があります。

図16.1 バグ修正のバックポートと更新の図



重要

OpenShift の開発には、リグレッションを防ぐ厳格なバックポートポリシーがあります。たとえば、バグは 4.15.z で修正する前に 4.16.z で修正する必要があります。そのため、更新の図では、マイナーバージョンが大きくても、時系列的に古いリリースへの更新 (4.15.24 から 4.16.2 への更新など) は許可されていません。

関連情報

- 更新チャネルとリリースについて

16.1.2.3. ターゲットリリースの選択

Red Hat OpenShift Container Platform Update Graph または cincinnati-graph-data リポジトリー を使用して、どのリリースに更新するかを決定します。

16.1.2.3.1. 利用可能な z-stream 更新の確認

新しい z-stream リリースに更新する前に、利用可能なバージョンを確認する必要があります。

注記

z-stream 更新を実行するときにチャネルを変更する必要はありません。

手順

1. 利用可能な z-stream リリースを確認します。以下のコマンドを実行します。

```
$ oc adm upgrade
```

出力例

```
Cluster version is 4.14.34

Upstream is unset, so the cluster will use an appropriate default.
Channel: stable-4.14 (available channels: candidate-4.14, candidate-4.15, eus-4.14, eus-4.16,
fast-4.14, fast-4.15, stable-4.14, stable-4.15)

Recommended updates:

  VERSION     IMAGE
  4.14.37     quay.io/openshift-release-dev/ocp-
release@sha256:14e6ba3975e6c73b659fa55af25084b20ab38a543772ca70e184b903db73092b
  4.14.36     quay.io/openshift-release-dev/ocp-
release@sha256:4bc4925e8028158e3f313aa83e59e181c94d88b4aa82a3b00202d6f354e8dfed
  4.14.35     quay.io/openshift-release-dev/ocp-
release@sha256:883088e3e6efa7443b0ac28cd7682c2fdbda889b576edad626769bf956ac0858
```

16.1.2.3.2. コントロールプレーンのみの更新のチャネルを変更する

コントロールプレーンのみの更新では、チャネルを必要なバージョンに変更する必要があります。

注記

z-stream 更新を実行するときにチャネルを変更する必要はありません。

手順

1. 現在設定されている更新チャネルを特定します。

```
$ oc get clusterversion -o=jsonpath='{.items[*].spec}' | jq
```

出力例

```
{
  "channel": "stable-4.14",
  "clusterID": "01eb9a57-2bfb-4f50-9d37-dc04bd5bac75"
}
```

2. 更新する新しいチャネルを参照するようにチャネルを変更します。

```
$ oc adm upgrade channel eus-4.16
```

3. 更新されたチャネルを確認します。

```
$ oc get clusterversion -o=jsonpath='{.items[*].spec}' | jq
```

出力例

```
{
  "channel": "eus-4.16",
  "clusterID": "01eb9a57-2bfb-4f50-9d37-dc04bd5bac75"
}
```

16.1.2.3.2.1. EUS から EUS への更新を早期に行うためにチャネルを変更する

OpenShift Container Platform の最新リリースへの更新パスは、マイナーリリースの最初の GA から 45 - 90 日が経過するまで、EUS チャネルでも stable チャネルでも利用できません。

新しいリリースへの更新のテストを開始する場合は、fast チャネルを使用できます。

手順

1. チャネルを **fast-<y+1>** に変更します。たとえば、以下のコマンドを実行します。

```
$ oc adm upgrade channel fast-4.16
```

2. 新しいチャネルの更新パスを確認します。以下のコマンドを実行します。

```
$ oc adm upgrade
```

```
Cluster version is 4.15.33

Upgradeable=False

  Reason: AdminAckRequired
  Message: Kubernetes 1.28 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/6958394 for details and instructions.

Upstream is unset, so the cluster will use an appropriate default.
Channel: fast-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.15, eus-4.16, fast-4.15, fast-4.16, stable-4.15, stable-4.16)

Recommended updates:

  VERSION     IMAGE
  4.16.14     quay.io/openshift-release-dev/ocp-release@sha256:6618dd3c0f5
  4.16.13     quay.io/openshift-release-dev/ocp-release@sha256:7a72abc3
  4.16.12     quay.io/openshift-release-dev/ocp-release@sha256:1c8359fc2
  4.16.11     quay.io/openshift-release-dev/ocp-release@sha256:bc9006febfe
  4.16.10     quay.io/openshift-release-dev/ocp-release@sha256:dece7b61b1
  4.15.36     quay.io/openshift-release-dev/ocp-release@sha256:c31a56d19
  4.15.35     quay.io/openshift-release-dev/ocp-release@sha256:f21253
  4.15.34     quay.io/openshift-release-dev/ocp-release@sha256:2dd69c5
```

3. 更新手順に従ってバージョン 4.16 にアップグレードします (バージョン 4.15 から <y+1>)。

注記

fast チャネルを使用している場合でも、EUS リリース間でワーカーノードを一時停止したままにすることができます。

4. 必要な <y+1> リリースに更新したら、チャネルを再度変更し、今度は **fast-<y+2>** に変更します。

5. EUS 更新手順に従って、必要な <y+2> リリースに更新します。

16.1.2.3.3. y-stream 更新のチャネルの変更

y-stream 更新では、チャネルを次のリリースチャネルに変更します。

注記

実稼働クラスターには、stable または EUS リリースチャネルを使用してください。

手順

1. 更新チャネルを変更します。

```
$ oc adm upgrade channel stable-4.15
```

2. 新しいチャネルの更新パスを確認します。以下のコマンドを実行します。

```
$ oc adm upgrade
```

出力例

```
Cluster version is 4.14.34

Upgradeable=False

  Reason: AdminAckRequired
  Message: Kubernetes 1.27 and therefore OpenShift 4.15 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/6958394 for details and instructions.

Upstream is unset, so the cluster will use an appropriate default.
Channel: stable-4.15 (available channels: candidate-4.14, candidate-4.15, eus-4.14, eus-4.15, fast-4.14, fast-4.15, stable-4.14, stable-4.15)

Recommended updates:

  VERSION     IMAGE
  4.15.33     quay.io/openshift-release-dev/ocp-release@sha256:7142dd4b560
  4.15.32     quay.io/openshift-release-dev/ocp-release@sha256:cda8ea5b13dc9
  4.15.31     quay.io/openshift-release-dev/ocp-release@sha256:07cf61e67d3eeee
  4.15.30     quay.io/openshift-release-dev/ocp-release@sha256:6618dd3c0f5
  4.15.29     quay.io/openshift-release-dev/ocp-release@sha256:7a72abc3
  4.15.28     quay.io/openshift-release-dev/ocp-release@sha256:1c8359fc2
  4.15.27     quay.io/openshift-release-dev/ocp-release@sha256:bc9006febfe
  4.15.26     quay.io/openshift-release-dev/ocp-release@sha256:dece7b61b1
```

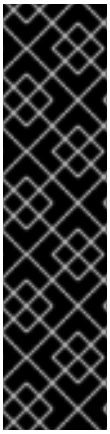
```
4.14.38    quay.io/openshift-release-dev/ocp-release@sha256:c93914c62d7
4.14.37    quay.io/openshift-release-dev/ocp-release@sha256:c31a56d19
4.14.36    quay.io/openshift-release-dev/ocp-release@sha256:f21253
4.14.35    quay.io/openshift-release-dev/ocp-release@sha256:2dd69c5
```

16.1.3. 通信事業者コアクラスタープラットフォームの更新準備

通常、通信事業者のクラスターはベアメタルハードウェア上で実行されます。多くの場合、重要なセキュリティー修正を適用したり、新しい機能を導入したり、OpenShift Container Platform の新しいリリースとの互換性を維持したりするために、ファームウェアを更新する必要があります。

16.1.3.1. ホストファームウェアが更新と互換性があることを確認する

お客様がクラスターで実行するファームウェアバージョンの更新は、お客様の責任です。ホストファームウェアの更新は、OpenShift Container Platform の更新プロセスには含まれません。OpenShift Container Platform のバージョンと一緒にファームウェアを更新することは推奨されません。



重要

ハードウェアベンダーは、ご使用のハードウェアに最新の認定済みファームウェアバージョンを適用することを推奨しています。通信事業者のユースケースでは、ファームウェアの更新を実稼働環境に適用する前に、必ずテスト環境で検証してください。ホストファームウェアが最適なものでないと、通信事業者の CNF ワークロードの高スループット特性に悪影響が及ぶ可能性があります。

新しいファームウェア更新を徹底的にテストして、OpenShift Container Platform の最新バージョンで期待どおりに動作することを確認してください。ターゲットの OpenShift Container Platform 更新バージョンを使用して最新のファームウェアバージョンをテストするのが理想的です。

16.1.3.2. レイヤード製品が更新と互換性があることを確認する

更新を開始する前に、更新先の OpenShift Container Platform のバージョンですべてのレイヤード製品が動作することを確認します。通常、これにはすべての Operator が含まれます。

手順

1. クラスターに現在インストールされている Operator を確認します。たとえば、以下のコマンドを実行します。

```
$ oc get csv -A
```

出力例

```
NAMESPACE                                          NAME            DISPLAY          VERSION   REPLACES
PHASE
gitlab-operator-kubernetes.v0.17.2     GitLab                    0.17.2    gitlab-operator-
kubernetes.v0.17.1    Succeeded
openshift-operator-lifecycle-manager   packageserver   Package Server   0.19.0
Succeeded
```

2. OLM を使用してインストールする Operator が更新先のバージョンと互換性があることを確認します。

Operator Lifecycle Manager (OLM) によってインストールされる Operator は、標準のクラスター Operator セットの一部ではありません。

Operator Update Information Checker を使用して、それぞれの y-stream 更新後に Operator を更新する必要があるかどうか、または次の EUS リリースに完全に更新されるまで待つことができるかどうかを確認してください。

ヒント

Operator Update Information Checker を使用して、Operator の特定のリリースと互換性のある OpenShift Container Platform のバージョンを確認することもできます。

3. OLM 以外の方法でインストールする Operator が更新先のバージョンと互換性があることを確認します。
Red Hat によって直接サポートされていない、OLM によってインストールされるすべての Operator について、Operator ベンダーに問い合わせてリリースの互換性を確認してください。

    - 一部の Operator は、OpenShift Container Platform の複数のリリースと互換性があります。クラスターの更新が完了するまで、Operator を更新する必要がない場合があります。詳細は、「ワーカーノードの更新」を参照してください。
    - 最初の y-stream コントロールプレーンの更新を実行した後に Operator を更新する方法は、「すべての OLM Operator の更新」を参照してください。

関連情報

- ワーカーノードの更新
- すべての OLM Operator の更新

16.1.3.3. 更新前にノードに MachineConfigPool ラベルを適用する

約 8 - 10 個のノードをグループ化するために、**MachineConfigPool** (**mcp**) ノードラベルを準備します。**mcp** グループを使用すると、クラスターの他の部分とは別にノードのグループを再起動できます。

mcp ノードラベルを使用すると、更新プロセス中にノードセットを一時停止および一時停止解除して、任意のタイミングで更新と再起動を実行できます。

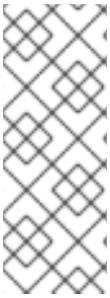
16.1.3.3.1. クラスター更新の段階的実施

更新中に問題が発生する場合があります。多くの場合、この問題はハードウェア障害やノードをリセットする必要性に関連しています。**mcp** ノードラベルを使用すると、重要なタイミングで更新を一時停止し、一時停止および一時停止解除したノードを追跡しながら、段階的にノードを更新できます。問題が発生した場合は、一時停止状態でないノードを使用して、すべてのアプリケーション Pod を実行し続けるのに十分な数のノードが実行されていることを確認します。

16.1.3.3.2. ワーカーノードを MachineConfigPool グループに分割する

ワーカーノードを **mcp** グループに分割する方法は、クラスター内のノードの数やノードロールに割り当てるノードの数によって異なります。デフォルトでは、クラスター内の 2 つのロールは、コントロールプレーンとワーカーです。

通信事業者のワークロードを実行するクラスターでは、ワーカーノードを CNF コントロールプレーンのロールと CNF データプレーンのロール間でさらに分割できます。ワーカーノードをこれら 2 つのグループに分割する **mcp** ロールラベルを追加します。



注記

大規模なクラスターでは、CNF コントロールプレーンのロールに 100 個ほどのワーカーノードが含まれる場合があります。クラスター内のノード数に関係なく、各 **MachineConfigPool** グループのノード数は、10 程度に抑えてください。これにより、一度に停止するノードの数を制御できます。複数の **MachineConfigPool** グループを使用すると、同時に複数のグループを一時停止解除して更新を高速化したり、更新を 2 つ以上のメンテナンス期間に分けたりすることができます。

15 個のワーカーノードを持つクラスターの例

15 個のワーカーノードを持つクラスターを考えてみます。

- 10 個のワーカーノードは CNF コントロールプレーンノードです。
- 5 個のワーカーノードは CNF データプレーンノードです。

CNF コントロールプレーンのロールとデータプレーンワーカーノードのロールを、それぞれ少なくとも 2 つの **mcp** グループに分割します。ロールごとに 2 つの **mcp** グループを用意することで、更新の影響を受けないノードのセットを 1 つ確保できます。

6 つのワーカーノードを持つクラスターの例

6 つのワーカーノードを持つクラスターを考えてみましょう。

- ワーカーノードを、それぞれ 2 つのノードからなる 3 つの **mcp** グループに分割します。

mcp グループの 1 つをアップグレードします。CNF の互換性を検証できるように、更新したノードを 1 日間そのままにしておきます。その後、他の 4 つのノードの更新を完了します。

重要

mcp グループの一時停止を解除するプロセスとペースは、CNF アプリケーションと設定によって決まります。

CNF Pod がクラスター内のノード間のスケジューリングに対応できる場合は、一度に複数の **mcp** グループの一時停止を解除し、**mcp** カスタムリソース (CR) の **MaxUnavailable** を最大 50% に設定できます。これにより、 **mcp** グループ内の最大半数のノードを再起動して更新できます。

16.1.3.3.3. 設定されているクラスター MachineConfigPool ロールの確認

クラスター内で現在設定されている **MachineConfigPool** ロールを確認します。

手順

1. クラスター内で現在設定されている **mcp** グループを取得します。

```
$ oc get mcp
```

出力例

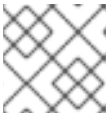
```
NAME    CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
master  rendered-master-bere83  True     False     False     3          3              3
0                  25d
worker  rendered-worker-245c4f  True     False     False     2          2              2
0                  25d
```

2. **mcp** ロールのリストをクラスター内のノードのリストと比較します。

```
$ oc get nodes
```

出力例

```
NAME         STATUS   ROLES                AGE   VERSION
ctrl-plane-0  Ready    control-plane,master  39d   v1.27.15+6147456
ctrl-plane-1  Ready    control-plane,master  39d   v1.27.15+6147456
ctrl-plane-2  Ready    control-plane,master  39d   v1.27.15+6147456
worker-0      Ready    worker               39d   v1.27.15+6147456
worker-1      Ready    worker               39d   v1.27.15+6147456
```



注記

mcp グループの変更を適用すると、ノードロールが更新されます。

ワーカーノードを **mcp** グループに分割する方法を決定します。

16.1.3.3.4. クラスターの MachineConfigPool グループの作成

mcp グループの作成は 2 つのステップで行います。

1. クラスター内のノードに **mcp** ラベルを追加します。
2. ラベルに基づいてノードをまとめるクラスターに **mcp** CR を適用します。

手順

1. ノードにラベルを付けて、**mcp** グループに配置できるようにします。以下のコマンドを実行します。

```
$ oc label node worker-0 node-role.kubernetes.io/mcp-1=
```

```
$ oc label node worker-1 node-role.kubernetes.io/mcp-2=
```

mcp-1 および **mcp-2** ラベルをノードに適用します。以下に例を示します。

出力例

```
NAME         STATUS   ROLES                AGE   VERSION
ctrl-plane-0  Ready    control-plane,master  39d   v1.27.15+6147456
```

```
ctrl-plane-1   Ready    control-plane,master   39d   v1.27.15+6147456
ctrl-plane-2   Ready    control-plane,master   39d   v1.27.15+6147456
worker-0       Ready    mcp-1,worker           39d   v1.27.15+6147456
worker-1       Ready    mcp-2,worker           39d   v1.27.15+6147456
```

2. クラスターの **mcp** CR としてラベルを適用する YAML カスタムリソース (CR) を作成します。次の YAML を **mcps.yaml** ファイルに保存します。

```
---
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-2
spec:
  machineConfigSelector:
    matchExpressions:
      - {
        key: machineconfiguration.openshift.io/role,
        operator: In,
        values: [worker,mcp-2]
       }
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-2: ""
---
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-1
spec:
  machineConfigSelector:
    matchExpressions:
      - {
        key: machineconfiguration.openshift.io/role,
        operator: In,
        values: [worker,mcp-1]
       }
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-1: ""
```

3. **MachineConfigPool** リソースを作成します。

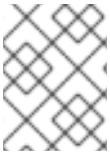
```
$ oc apply -f mcps.yaml
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/mcp-2 created
```

検証

クラスターに **MachineConfigPool** リソースが適用される様子を監視します。**mcp** リソースを適用すると、ノードが新しいマシン設定プールに追加されます。これには数分かかります。



注記

ノードは、**mcp** グループに追加されている間は再起動しません。元のワーカーおよびマスター **mcp** グループは変更されません。

- 新しい **mcp** リソースのステータスを確認します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG              UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
master   rendered-master-be3e83   True      False      False      3
3                   3                     0                   25d
mcp-1    rendered-mcp-1-2f4c4f    False     True       True       1
0                   0                     0                   10s
mcp-2    rendered-mcp-2-2r4s1f    False     True       True       1
0                   0                     0                   10s
worker   rendered-worker-23fc4f   False     True       True       0
0                   0                     2                   25d
```

最終的に、リソースは完全に適用されます。

```
NAME     CONFIG              UPDATED   UPDATING   DEGRADED   MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
master   rendered-master-be3e83   True      False      False      3
3                   3                     0                   25d
mcp-1    rendered-mcp-1-2f4c4f    True      False      False      1
1                   1                     0                   7m33s
mcp-2    rendered-mcp-2-2r4s1f    True      False      False      1
1                   1                     0                   51s
worker   rendered-worker-23fc4f   True      False      False      0
0                   0                     0                   25d
```

関連情報

- コントロールプレーンのみの更新を実行する
- 更新期間に影響する要因
- Pod Disruption Budget により CNF ワークロードを無停止で実行する
- Pod が同じクラスターノード上で実行されないようにする

16.1.3.4. 通信事業者のデプロイメント環境に関する考慮事項

通信事業者の環境では、ほとんどのクラスターはインターネット非接続のネットワーク内にあります。このような環境でクラスターを更新するには、オフラインイメージリポジトリーを更新する必要があります。

関連情報

- API 互換性ガイドライン
- oc-mirror プラグイン v2 を使用した非接続インストールのイメージのミラーリング

16.1.3.5. クラスタープラットフォームの更新の準備

クラスターを更新する前に、いくつかの基本的なチェックと検証を実行して、クラスターの更新の準備ができていることを確認します。

手順

1. 次のコマンドを実行して、クラスター内に失敗した Pod や進行中の Pod がないことを確認します。

```
$ oc get pods -A | grep -E -vi 'complete|running'
```

注記

保留中の状態の Pod がある場合は、このコマンドを複数回実行する必要がある場合があります。

2. クラスター内のすべてのノードが利用可能であることを確認します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                AGE   VERSION
ctrl-plane-0  Ready    control-plane,master   32d   v1.27.15+6147456
ctrl-plane-1  Ready    control-plane,master   32d   v1.27.15+6147456
ctrl-plane-2  Ready    control-plane,master   32d   v1.27.15+6147456
worker-0      Ready    mcp-1,worker         32d   v1.27.15+6147456
worker-1      Ready    mcp-2,worker         32d   v1.27.15+6147456
```

3. すべてのベアメタルノードがプロビジョニングされ、準備ができていることを確認します。

```
$ oc get bmh -n openshift-machine-api
```

出力例

```
NAME          STATE        CONSUMER                ONLINE   ERROR   AGE
ctrl-plane-0  unmanaged    cnf-58879-master-0        true          33d
ctrl-plane-1  unmanaged    cnf-58879-master-1        true          33d
ctrl-plane-2  unmanaged    cnf-58879-master-2        true          33d
worker-0      unmanaged    cnf-58879-worker-0-45879  true           33d
worker-1      progressing  cnf-58879-worker-0-dszsh  false          1d (1)
```

(1) **worker-1** ノードのプロビジョニング中にエラーが発生しています。

検証

検証

- すべてのクラスター Operator の準備ができていることを確認します。

```
$ oc get co
```

出力例

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
SINCE   MESSAGE
authentication                             4.14.34   True        False         False     17h
baremetal                                  4.14.34   True        False         False     32d

...

service-ca                                 4.14.34   True        False         False     32d
storage                                    4.14.34   True        False         False     32d
```

関連情報

- Pod の問題の調査

16.1.4. 通信事業者コア CNF クラスター更新前の CNF Pod の設定

クラウドネイティブネットワーク機能 (CNF) を開発するときは、Red Hat best practices for Kubernetes のガイダンスに従って、更新中にクラスターが Pod をスケジュールできるようにします。

重要

Deployment リソースを使用して、常に Pod をグループでデプロイしてください。**Deployment** リソースは、単一障害点が生まれないように、利用可能なすべての Pod にワークロードを分散します。**Deployment** リソースによって管理されている Pod が削除されると、新しい Pod が自動的にその代わりになります。

関連情報

- Red Hat best practices for Kubernetes

16.1.4.1. Pod Disruption Budget により CNF ワークロードを無停止で実行する

適用する **PodDisruptionBudget** カスタムリソース (CR) で Pod Disruption Budget を設定すると、デプロイメント内の Pod の最小数を設定して、CNF ワークロードを無停止で実行できます。この値を設定するときは注意してください。不適切に設定すると更新が失敗する可能性があります。

たとえば、デプロイメントに 4 つの Pod があり、Pod Disruption Budget を 4 に設定した場合、クラスタースケジューラーによって常に 4 つの Pod が実行され、Pod をスケールダウンできなくなります。

そうではなく、Pod Disruption Budget を 2 に設定し、4 つの Pod のうち 2 つをダウン状態としてスケジュールできるようにします。そうすれば、それらの Pod が配置されているワーカーノードを再起動できます。

注記

Pod Disruption Budget を 2 に設定しても、更新中など、一定期間にわたりデプロイメントが 2 つの Pod のみで実行されるわけではありません。クラススケジューラーは、2 つの古い Pod を置き換える 2 つの新しい Pod を作成します。しかし、新しい Pod がオンラインになってから古い Pod が削除されるまでには、少し時間がかかります。

関連情報

- Pod の Disruption Budget を使用して起動している Pod 数の指定
- Pod プリエンプションおよび他のスケジューラーの設定

16.1.4.2. Pod が同じクラスターノード上で実行されないようにする

Kubernetes で高可用性を実現するには、クラスター内の別々のノードで重複したプロセスを実行する必要があります。これにより、1 つのノードが使用できなくなった場合でも、アプリケーションを引き続き実行できます。OpenShift Container Platform では、デプロイメント内の別々の Pod にプロセスを自動的に複製できます。デプロイメント内の Pod が同じクラスターノード上で実行されないようにするには、**Pod** 仕様でアンチアフィニティーを設定します。

更新時に Pod のアンチアフィニティーを設定すると、Pod がクラスター内のノード全体に均等に分散されるようになります。つまり、更新時にノードの再起動が容易になります。たとえば、ノード上に 1 つのデプロイメントの Pod が 4 つあり、Pod Disruption Budget が Pod を一度に 1 つだけ削除できるように設定されている場合、そのノードの再起動には 4 倍の時間がかかります。Pod のアンチアフィニティーを設定すると、Pod がクラスター全体に分散され、このような事態の発生が防止されます。

関連情報

- Pod アフィニティールールの設定

16.1.4.3. アプリケーションの liveness、readiness、および startup プローブ

更新をスケジュールする前に、liveness、readiness、および startup プローブを使用して、ライブアプリケーションコンテナーの健全性をチェックできます。これらは、アプリケーションコンテナーの状態の保持に依存する Pod で使用するのに非常に便利なツールです。

liveness ヘルスチェック

コンテナーが実行中かどうかを確認します。コンテナーの liveness プローブが失敗した場合、Pod が再起動ポリシーに基づいて応答します。

readiness プローブ

コンテナーがサービス要求を受け入れる準備ができているかどうかを確認します。コンテナーの readiness プローブが失敗した場合、kubelet が利用可能なサービスエンドポイントのリストからコンテナーを削除します。

startup プローブ

startup プローブは、コンテナー内のアプリケーションが起動しているかどうかを示します。その他のプローブはすべて、起動に成功するまで無効にされます。startup プローブが成功しない場合、kubelet がコンテナーを強制終了し、そのコンテナーが Pod の **restartPolicy** 設定の対象となります。

関連情報

- ヘルスチェックについて

16.1.5. 通信事業者コア CNF クラスターを更新する前の作業

クラスターの更新を開始する前に、ワーカーノードを一時停止し、etcd データベースをバックアップし、最終的なクラスターのヘルスチェックを実行する必要があります。

16.1.5.1. 更新前のワーカーノードの一時停止

更新を開始する前に、ワーカーノードを一時停止する必要があります。次の例では、**mcp-1** と **mcp-2** という 2 つの **mcp** グループがあります。パッチを適用して、これらの各 **MachineConfigPool** グループの **spec.paused** フィールドを **true** にします。

手順

1. 次のコマンドを実行して、**mcp** CR にパッチを適用し、ノードを一時停止し、それらのノードから Pod をドレインして削除します。

```
$ oc patch mcp/mcp-1 --type merge --patch '{"spec":{"paused":true}}'
```

```
$ oc patch mcp/mcp-2 --type merge --patch '{"spec":{"paused":true}}'
```

2. 一時停止した **mcp** グループのステータスを取得します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   true
mcp-2   true
```

注記

デフォルトのコントロールプレーンおよびワーカーの **mcp** グループは、更新中に変更されません。

16.1.5.2. 更新を開始する前の etcd データベースのバックアップ

更新を開始する前に、etcd データベースをバックアップする必要があります。

16.1.5.2.1. etcd データのバックアップ

以下の手順に従って、etcd スナップショットを作成し、静的 Pod のリソースをバックアップして etcd データをバックアップします。このバックアップは保存でき、etcd を復元する必要がある場合に後で使用することができます。

重要

単一のコントロールプレーンホストからのバックアップのみを保存します。クラスター内の各コントロールプレーンホストからのバックアップは取得しないでください。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- クラスター全体のプロキシーが有効になっているかどうかを確認している。

ヒント

oc get proxy cluster -o yaml の出力を確認して、プロキシーが有効にされているかどうかを確認できます。プロキシーは、**httpProxy**、**httpsProxy**、および **noProxy** フィールドに値が設定されている場合に有効にされます。

手順

1. コントロールプレーンノードの root としてデバッグセッションを開始します。

```
$ oc debug --as-root node/<node_name>
```

2. デバッグシェルで root ディレクトリーを **/host** に変更します。

```
sh-4.4# chroot /host
```

3. クラスター全体のプロキシーが有効になっている場合は、次のコマンドを実行して、**NO_PROXY**、**HTTP_PROXY**、および **HTTPS_PROXY** 環境変数をエクスポートします。

```
$ export HTTP_PROXY=http://<your_proxy.example.com>:8080
```

```
$ export HTTPS_PROXY=https://<your_proxy.example.com>:8080
```

```
$ export NO_PROXY=<example.com>
```

4. デバッグシェルで **cluster-backup.sh** スクリプトを実行し、バックアップの保存先となる場所を渡します。

ヒント

cluster-backup.sh スクリプトは etcd Cluster Operator のコンポーネントとして維持され、**etcdctl snapshot save** コマンドに関連するラッパーです。

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

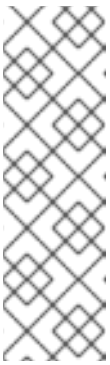
スクリプトの出力例

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
```

```
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

この例では、コントロールプレーンホストの **/home/core/assets/backup/** ディレクトリーにファイルが 2 つ作成されます。

- **snapshot_<datetimestamp>.db**: このファイルは etcd スナップショットです。**cluster-backup.sh** スクリプトで、その有効性を確認します。
- **static_kuberesources_<datetimestamp>.tar.gz**: このファイルには、静的 Pod のリソースが含まれます。etcd 暗号化が有効にされている場合、etcd スナップショットの暗号化キーも含まれます。



注記

etcd 暗号化が有効にされている場合、セキュリティー上の理由から、この 2 つ目のファイルを etcd スナップショットとは別に保存することが推奨されます。ただし、このファイルは etcd スナップショットから復元するために必要になります。

etcd 暗号化はキーではなく値のみを暗号化することに注意してください。つまり、リソースタイプ、namespace、およびオブジェクト名は暗号化されません。

16.1.5.2.2. シングル etcd バックアップの作成

次の手順でカスタムリソース (CR) を作成して適用することで、シングル etcd バックアップを作成します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) にアクセスできる。

手順

- 動的にプロビジョニングされたストレージが利用可能な場合は、次の手順を実行して、単一の自動 etcd バックアップを作成します。

a. 次の例のような内容で、**etcd-backup-pvc.yaml** という名前の永続ボリューム要求 (PVC) を作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 200Gi (1)
  volumeMode: Filesystem
```

(1) PVC に利用できるストレージの量。この値は、要件に合わせて調整します。

b. 以下のコマンドを実行して PVC を適用します。

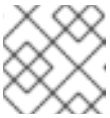
```
$ oc apply -f etcd-backup-pvc.yaml
```

c. 次のコマンドを実行して、PVC が作成されたことを確認します。

```
$ oc get pvc
```

出力例

```
NAME              STATUS    VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
etcd-backup-pvc   Bound                                      51s
```



注記

動的 PVC は、マウントされるまで **Pending** 状態から遷移しません。

d. 次の例のような内容で、**etcd-single-backup.yaml** という名前の CR ファイルを作成します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc (1)
```

(1) バックアップを保存する PVC の名前。この値は、使用している環境に応じて調整してください。

e. CR を適用してシングルバックアップを開始します。

```
$ oc apply -f etcd-single-backup.yaml
```

- 動的にプロビジョニングされたストレージが利用できない場合は、次の手順を実行して、単一の自動 etcd バックアップを作成します。

  a. 次の内容で、**etcd-backup-local-storage.yaml** という名前の **StorageClass** CR ファイルを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: etcd-backup-local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

  b. 次のコマンドを実行して、**StorageClass** CR を適用します。

```
$ oc apply -f etcd-backup-local-storage.yaml
```

  c. 次の例のような内容の **etcd-backup-pv-fs.yaml** という名前の PV を作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-backup-pv-fs
spec:
  capacity:
    storage: 100Gi 1
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: etcd-backup-local-storage
  local:
    path: /mnt
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - <example_master_node> 2
```

  1 PV が使用できるストレージの量。この値は、要件に合わせて調整します。

  2 この値は、この PV をアタッチするノードに置き換えます。

  d. 次のコマンドを実行して、PV が作成されたことを確認します。

```
$ oc get pv
```

出力例

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM   STORAGECLASS            REASON   AGE
etcd-backup-pv-fs   100Gi      RWO            Retain           Available           etcd-backup-
local-storage           10s
```

e. 次の例のような内容で、**etcd-backup-pvc.yaml** という名前の PVC を作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: etcd-backup-pvc
  namespace: openshift-etcd
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi 1
```

1 PVC に利用できるストレージの量。この値は、要件に合わせて調整します。

f. 以下のコマンドを実行して PVC を適用します。

```
$ oc apply -f etcd-backup-pvc.yaml
```

g. 次の例のような内容で、**etcd-single-backup.yaml** という名前の CR ファイルを作成します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: EtcdBackup
metadata:
  name: etcd-single-backup
  namespace: openshift-etcd
spec:
  pvcName: etcd-backup-pvc 1
```

1 バックアップを保存する永続ボリューム要求 (PVC) の名前。この値は、使用している環境に応じて調整してください。

h. CR を適用してシングルバックアップを開始します。

```
$ oc apply -f etcd-single-backup.yaml
```

関連情報

- etcd のバックアップ

16.1.5.3. クラスターの健全性の確認

更新時には、クラスターの健全性を頻繁に確認する必要があります。ノードのステータス、クラスター Operator のステータス、失敗した Pod を確認します。

手順

1. 次のコマンドを実行して、クラスター Operator のステータスを確認します。

```
$ oc get co
```

出力例

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
SINCE   MESSAGE
authentication                             4.14.34   True        False         False      4d22h
baremetal                                  4.14.34   True        False         False      4d22h
cloud-controller-manager                   4.14.34   True        False         False      4d23h
cloud-credential                           4.14.34   True        False         False      4d23h
cluster-autoscaler                         4.14.34   True        False         False      4d22h
config-operator                            4.14.34   True        False         False      4d22h
console                                    4.14.34   True        False         False      4d22h
...
service-ca                                 4.14.34   True        False         False      4d22h
storage                                    4.14.34   True        False         False      4d22h
```

2. クラスターノードのステータスを確認します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                  AGE     VERSION
ctrl-plane-0  Ready    control-plane,master   4d22h   v1.27.15+6147456
ctrl-plane-1  Ready    control-plane,master   4d22h   v1.27.15+6147456
ctrl-plane-2  Ready    control-plane,master   4d22h   v1.27.15+6147456
worker-0      Ready    mcp-1,worker           4d22h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker           4d22h   v1.27.15+6147456
```

3. 進行中の Pod や失敗した Pod がないことを確認します。次のコマンドを実行したときに、Pod が 1 つも返されないようにします。

```
$ oc get po -A | grep -E -iv 'running|complete'
```

16.1.6. コントロールプレーンのみのクラスター更新の完了

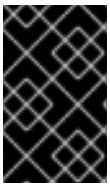
次の手順に従って、コントロールプレーンのみのクラスター更新を実行し、更新が完了するまで監視します。

重要

コントロールプレーンのみの更新は、以前は EUS から EUS への更新と呼ばれていました。コントロールプレーンのみの更新は、OpenShift Container Platform の偶数番号のマイナーバージョン間でのみ実行可能です。

16.1.6.1. コントロールプレーンのみまたは y-stream の更新の承認

4.11 以降のすべてのバージョンに更新する場合は、更新の続行を手動で承認する必要があります。



重要

更新を承認する前に、更新先のバージョンで削除されている Kubernetes API を使用していないことを確認してください。たとえば、OpenShift Container Platform 4.17 では、API の削除はありません。詳細は、「Kubernetes API の削除」を参照してください。

前提条件

- クラスターで実行されているすべてのアプリケーションの API が、OpenShift Container Platform の次期 y-stream リリースと互換性があることを確認した。互換性の詳細は、「更新バージョン間のクラスター API バージョンの検証」を参照してください。

手順

- 次のコマンドを実行して、管理者としての承認を完了し、クラスターの更新を開始します。

```
$ oc adm upgrade
```

クラスターの更新が正常に完了しない場合は、**Reason** および **Message** セクションに更新の失敗に関する詳細が表示されます。

出力例

```
Cluster version is 4.15.45

Upgradeable=False

  Reason: MultipleReasons
  Message: Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"

ReleaseAccepted=False

  Reason: PreconditionChecks
  Message: Preconditions failed for payload loaded version="4.16.34" image="quay.io/openshift-release-dev/ocp-release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d520b353b8": Precondition "ClusterVersionUpgradeable" failed because of "MultipleReasons": Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"
```

```
Upstream is unset, so the cluster will use an appropriate default.
Channel: eus-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.16, fast-4.15,
fast-4.16, stable-4.15, stable-4.16)

Recommended updates:

  VERSION     IMAGE
  4.16.34     quay.io/openshift-release-dev/ocp-
release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d520b353b
8
```

注記

この例は、Red Hat のナレッジベース記事 (Preparing to upgrade to OpenShift Container Platform 4.16) に、リリース間の API 互換性の検証に関する詳細が記載されています。

検証

- 次のコマンドを実行して更新を確認します。

```
$ oc get configmap admin-acks -n openshift-config -o json | jq .data
```

出力例

```
{
  "ack-4.14-kube-1.28-api-removals-in-4.15": "true",
  "ack-4.15-kube-1.29-api-removals-in-4.16": "true"
}
```

注記

この例では、コントロールプレーンのみの更新で、クラスターをバージョン 4.14 から 4.15 に更新してから、4.15 から 4.16 に更新します。

関連情報

- Kubernetes API の削除
- 更新バージョン間のクラスター API バージョンの検証

16.1.6.2. クラスターの更新の開始

ある y-stream リリースから次のリリースに更新する場合、中間の z-stream リリースにも互換性があることを確認する必要があります。

注記

oc adm upgrade コマンドを実行すると、更新先のリリースが実行可能なリリースであることを確認できます。**oc adm upgrade** コマンドは、互換性のある更新リリースをリスト表示します。

手順

1. 更新を開始します。

```
$ oc adm upgrade --to=4.15.33
```

重要

- **コントロールプレーンのみの更新**: 中間の <y+1> リリースパスを指定します。
- **y-stream 更新** - Kubernetes バージョンスキューポリシー に準拠した正しい <y.z> リリースを使用していることを確認します。
- **z-stream の更新** - 特定のリリースへの移行に問題がないことを確認します。

出力例

```
Requested update to 4.15.33 1
```

1 **Requested update** 値は、実際の更新に応じて変わります。

関連情報

- ターゲットリリースの選択

16.1.6.3. クラスターの更新の監視

更新時には、クラスターの健全性を頻繁に確認する必要があります。ノードのステータス、クラスター Operator のステータス、失敗した Pod を確認します。

手順

- クラスターの更新を監視します。たとえば、バージョン 4.14 から 4.15 へのクラスターの更新を監視するには、次のコマンドを実行します。

```
$ watch "oc get clusterversion; echo; oc get co | head -1; oc get co | grep 4.14; oc get co | grep 4.15; echo; oc get no; echo; oc get po -A | grep -E -iv 'running|complete'"
```

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.14.34   True        True          4m6s    Working towards 4.15.33: 111 of 873 done (12% complete), waiting on kube-apiserver

NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
MESSAGE
authentication                             4.14.34   True        False         False      4d22h
baremetal                                  4.14.34   True        False         False      4d23h
cloud-controller-manager                   4.14.34   True        False         False      4d23h
cloud-credential                           4.14.34   True        False         False      4d23h
cluster-autoscaler                         4.14.34   True        False         False      4d23h
```

```
console                    4.14.34   True       False       False     4d22h

...

storage                    4.14.34   True       False       False     4d23h
config-operator            4.15.33   True       False       False     4d23h
etcd                       4.15.33   True       False       False     4d23h

NAME          STATUS   ROLES                AGE     VERSION
ctrl-plane-0  Ready    control-plane,master  4d23h   v1.27.15+6147456
ctrl-plane-1  Ready    control-plane,master  4d23h   v1.27.15+6147456
ctrl-plane-2  Ready    control-plane,master  4d23h   v1.27.15+6147456
worker-0      Ready    mcp-1,worker          4d23h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker          4d23h   v1.27.15+6147456

NAMESPACE             NAME                     READY   STATUS              RESTARTS   AGE
openshift-marketplace   redhat-marketplace-rf86t   0/1     ContainerCreating   0          0s
```

検証

更新中、この **watch** コマンドは、一度に 1 つまたは複数のクラスター Operator を循環し、**MESSAGE** 列に Operator 更新のステータスを表示します。

クラスター Operator の更新プロセスが完了すると、各コントロールプレーンノードが 1 つずつ再起動します。

注記

更新のこの部分で、クラスター Operator が再度更新中であることやデグレード状態であることを示すメッセージが報告されます。これは、ノードを再起動している間、コントロールプレーンノードがオフラインになるためです。

最後のコントロールプレーンノードの再起動が完了するとすぐに、更新されたクラスターのバージョンが表示されます。

コントロールプレーンの更新が完了すると、次のようなメッセージが表示されます。この例では、中間の y-stream リリースへの更新が完了したことを示しています。

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.15.33   True        False         28m     Cluster version is 4.15.33

NAME                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
authentication             4.15.33   True        False         False      5d
baremetal                  4.15.33   True        False         False      5d
cloud-controller-manager   4.15.33   True        False         False      5d1h
cloud-credential           4.15.33   True        False         False      5d1h
cluster-autoscaler         4.15.33   True        False         False      5d
config-operator            4.15.33   True        False         False      5d
console                    4.15.33   True        False         False      5d

...

service-ca                 4.15.33   True        False         False      5d
storage                    4.15.33   True        False         False      5d
```

```
NAME          STATUS   ROLES                AGE   VERSION
ctrl-plane-0  Ready    control-plane,master  5d   v1.28.13+2ca1a23
ctrl-plane-1  Ready    control-plane,master  5d   v1.28.13+2ca1a23
ctrl-plane-2  Ready    control-plane,master  5d   v1.28.13+2ca1a23
worker-0      Ready    mcp-1,worker          5d   v1.28.13+2ca1a23
worker-1      Ready    mcp-2,worker          5d   v1.28.13+2ca1a23
```

16.1.6.4. OLM Operator の更新

通信事業者の環境では、ソフトウェアを実稼働クラスターにロードする前に検査する必要があります。また、実稼働クラスターが非接続ネットワーク内で設定されるため、必ずしもインターネットに直接接続されているわけではありません。クラスターが非接続ネットワーク内にあるため、新しいバージョンをクラスターごとに管理できるように、インストール時に OpenShift Operator を手動で更新するように設定します。Operator を新しいバージョンに移行するには、次の手順を実行します。

手順

1. 更新する必要がある Operator を確認します。

```
$ oc get installplan -A | grep -E 'APPROVED|false'
```

出力例

```
NAMESPACE          NAME          CSV                                        APPROVAL
APPROVED
metallb-system     install-nwjnh  metallb-operator.v4.16.0-202409202304      Manual
false
openshift-nmstate  install-5r7wr  kubernetes-nmstate-operator.4.16.0-202409251605
Manual     false
```

2. これらの Operator の **InstallPlan** リソースにパッチを適用します。

```
$ oc patch installplan -n metallb-system install-nwjnh --type merge --patch \
'{"spec":{"approved":true}}'
```

出力例

```
installplan.operators.coreos.com/install-nwjnh patched
```

3. 次のコマンドを実行して namespace を監視します。

```
$ oc get all -n metallb-system
```

出力例

```
NAME                                      READY   STATUS             RESTARTS   AGE
pod/metallb-operator-controller-manager-69b5f884c-8bp22   0/1     ContainerCreating   0
4s
pod/metallb-operator-controller-manager-77895bdb46-bqjdx   1/1     Running            0
4m1s
pod/metallb-operator-webhook-server-5d9b968896-vnbhk       0/1     ContainerCreating   0
4s
pod/metallb-operator-webhook-server-d76f9c6c8-57r4w        1/1     Running            0
```

```
4m1s

...

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/metallb-operator-controller-manager-69b5f884c    1         1         0       4s
replicaset.apps/metallb-operator-controller-manager-77895bdb46   1         1         1       4m1s
replicaset.apps/metallb-operator-controller-manager-99b76f88     0         0         0       4m40s
replicaset.apps/metallb-operator-webhook-server-5d9b968896       1         1         0       4s
replicaset.apps/metallb-operator-webhook-server-6f7dbfdb88       0         0         0       4m40s
replicaset.apps/metallb-operator-webhook-server-d76f9c6c8        1         1         1       4m1s
```

更新が完了すると、必要な Pod が **Running** 状態になり、必要な **ReplicaSet** リソースが準備完了になるはずです。

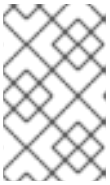
```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/metallb-operator-controller-manager-69b5f884c-8bp22   1/1     Running   0          25s
pod/metallb-operator-webhook-server-5d9b968896-vnbhk      1/1     Running   0          25s

...

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/metallb-operator-controller-manager-69b5f884c    1         1         1       25s
replicaset.apps/metallb-operator-controller-manager-77895bdb46   0         0         0       4m22s
replicaset.apps/metallb-operator-webhook-server-5d9b968896       1         1         1       25s
replicaset.apps/metallb-operator-webhook-server-d76f9c6c8        0         0         0       4m22s
```

検証

- Operator を再度更新する必要がないことを確認します。

```
$ oc get installplan -A | grep -E 'APPROVED|false'
```

出力は返されないはずです。



注記

Operator によっては、最終バージョンの前にインストールする必要がある中間の z-stream リリースバージョンがあるため、更新を 2 回承認する必要がある場合があります。

関連情報

- ワーカーノードの更新

16.1.6.4.1. 2 回目の y-stream 更新の実行

最初の y-stream 更新を完了したら、y-stream のコントロールプレーンのバージョンを新しい EUS バージョンに更新する必要があります。

手順

1. 選択した <4.y.z> リリースが、適切な移行先チャネルとして現在もリスト表示されることを確認します。

```
$ oc adm upgrade
```

出力例

```
Cluster version is 4.15.33

Upgradeable=False

  Reason: AdminAckRequired
  Message: Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.

Upstream is unset, so the cluster will use an appropriate default.
Channel: eus-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.16, fast-4.15, fast-4.16, stable-4.15, stable-4.16)

Recommended updates:

  VERSION     IMAGE
  4.16.14     quay.io/openshift-release-dev/ocp-release@sha256:0521a0f1acd2d1b77f76259cb9bae9c743c60c37d9903806a3372c1414253658
  4.16.13     quay.io/openshift-release-dev/ocp-release@sha256:6078cb4ae197b5b0c526910363b8aff540343bfac62ecb1ead9e068d541da27b
  4.15.34     quay.io/openshift-release-dev/ocp-release@sha256:f2e0c593f6ed81250c11d0bac94dbaf63656223477b7e8693a652f933056af6e
```

注記

新しい y-stream リリースの最初の GA 直後に更新すると、**oc adm upgrade** コマンドを実行したときに、利用可能な新しい y-stream リリースが表示されない場合があります。

2. オプション: 推奨されない更新リリースの候補を表示します。以下のコマンドを実行します。

```
$ oc adm upgrade --include-not-recommended
```

出力例

```
Cluster version is 4.15.33

Upgradeable=False

  Reason: AdminAckRequired
  Message: Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.

Upstream is unset, so the cluster will use an appropriate default.Channel: eus-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.16, fast-4.15, fast-4.16, stable-4.15, stable-4.16)
```

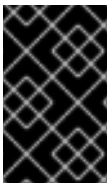
```
Recommended updates:

  VERSION     IMAGE
  4.16.14     quay.io/openshift-release-dev/ocp-release@sha256:0521a0f1acd2d1b77f76259cb9bae9c743c60c37d9903806a3372c1414253658
  4.16.13     quay.io/openshift-release-dev/ocp-release@sha256:6078cb4ae197b5b0c526910363b8aff540343bfac62ecb1ead9e068d541da27b
  4.15.34     quay.io/openshift-release-dev/ocp-release@sha256:f2e0c593f6ed81250c11d0bac94dbaf63656223477b7e8693a652f933056af6e


Supported but not recommended updates:

  Version: 4.16.15
  Image: quay.io/openshift-release-dev/ocp-release@sha256:671bc35e
  Recommended: Unknown
  Reason: EvaluationFailed
  Message: Exposure to AzureRegistryImagePreservation is unknown due to an evaluation failure: invalid PromQL result length must be one, but is 0
  In Azure clusters, the in-cluster image registry may fail to preserve images on update.
https://issues.redhat.com/browse/IR-461
```

注記

この例では、Microsoft Azure でホストされているクラスターに影響を与える可能性のある潜在的なエラーが示されています。ベアメタルクラスターのリスクは示されていません。

16.1.6.4.2. y-stream リリースの更新の承認

y-stream リリース間で移行する場合、更新を明示的に承認するために patch コマンドを実行する必要があります。**oc adm upgrade** コマンドの出力に、実行する特定のコマンドを示す URL が表示されます。



重要

更新を承認する前に、更新先のバージョンで削除されている Kubernetes API を使用していないことを確認してください。たとえば、OpenShift Container Platform 4.17 では、API の削除はありません。詳細は、「Kubernetes API の削除」を参照してください。

前提条件

- クラスターで実行されているすべてのアプリケーションの API が、OpenShift Container Platform の次期 y-stream リリースと互換性があることを確認した。互換性の詳細は、「更新バージョン間のクラスター API バージョンの検証」を参照してください。

手順

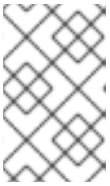
- 次のコマンドを実行して、管理者としての承認を完了し、クラスターの更新を開始します。

```
$ oc adm upgrade
```

クラスターの更新が正常に完了しない場合は、**Reason** および **Message** セクションに更新の失敗に関する詳細が表示されます。

出力例

```
Cluster version is 4.15.45

Upgradeable=False

  Reason: MultipleReasons
  Message: Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"

ReleaseAccepted=False

  Reason: PreconditionChecks
  Message: Preconditions failed for payload loaded version="4.16.34" image="quay.io/openshift-release-dev/ocp-release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d520b353b8": Precondition "ClusterVersionUpgradeable" failed because of "MultipleReasons": Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"

Upstream is unset, so the cluster will use an appropriate default.
Channel: eus-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.16, fast-4.15, fast-4.16, stable-4.15, stable-4.16)

Recommended updates:

  VERSION     IMAGE
  4.16.34     quay.io/openshift-release-dev/ocp-release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d520b353b8
```



注記

この例は、Red Hat のナレッジベース記事 (Preparing to upgrade to OpenShift Container Platform 4.16) に、リリース間の API 互換性の検証に関する詳細が記載されています。

検証

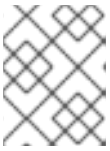
- 次のコマンドを実行して更新を確認します。

```
$ oc get configmap admin-acks -n openshift-config -o json | jq .data
```

-

出力例

```
{
  "ack-4.14-kube-1.28-api-removals-in-4.15": "true",
  "ack-4.15-kube-1.29-api-removals-in-4.16": "true"
}
```



注記

この例では、コントロールプレーンのみの更新で、クラスターをバージョン 4.14 から 4.15 に更新してから、4.15 から 4.16 に更新します。

関連情報

- OpenShift Container Platform 4.16 への更新の準備
- 更新バージョン間のクラスター API バージョンの検証

16.1.6.5. y-stream コントロールプレーンの更新の開始

移行先の完全な新しいリリースを決定したら、**oc adm upgrade –to=x.y.z** コマンドを実行できます。

手順

- y-stream コントロールプレーンの更新を開始します。たとえば、以下のコマンドを実行します。

```
$ oc adm upgrade --to=4.16.14
```

出力例

```
Requested update to 4.16.14
```

移行先の z-stream リリースに、使用中のプラットフォーム以外のプラットフォームに関する問題がある場合があります。次の例は、Microsoft Azure でのクラスター更新の潜在的な問題を示しています。

```
$ oc adm upgrade --to=4.16.15
```

出力例

```
error: the update 4.16.15 is not one of the recommended updates, but is available as a
conditional update. To accept the Recommended=Unknown risk and to proceed with update
use --allow-not-recommended.
  Reason: EvaluationFailed
  Message: Exposure to AzureRegistryImagePreservation is unknown due to an evaluation
failure: invalid PromQL result length must be one, but is 0
  In Azure clusters, the in-cluster image registry may fail to preserve images on update.
https://issues.redhat.com/browse/IR-461
```

注記

この例では、Microsoft Azure でホストされているクラスターに影響を与える可能性のある潜在的なエラーが示されています。ベアメタルクラスターのリスクは示されていません。

```
$ oc adm upgrade --to=4.16.15 --allow-not-recommended
```

出力例

```
warning: with --allow-not-recommended you have accepted the risks with 4.14.11 and
bypassed Recommended=Unknown EvaluationFailed: Exposure to
AzureRegistryImagePreservation is unknown due to an evaluation failure: invalid PromQL
result length must be one, but is 0
In Azure clusters, the in-cluster image registry may fail to preserve images on update.
https://issues.redhat.com/browse/IR-461

Requested update to 4.16.15
```

16.1.6.6. <y+1> クラスター更新の 2 番目の部分を監視する

クラスターの 2 番目の部分が <y+1> バージョンに更新される様子を監視します。

手順

- <y+1> 更新の 2 番目の部分の進行状況を監視します。たとえば、4.15 から 4.16 への更新を監視するには、次のコマンドを実行します。

```
$ watch "oc get clusterversion; echo; oc get co | head -1; oc get co | grep 4.15; oc get co |
grep 4.16; echo; oc get no; echo; oc get po -A | grep -E -iv 'running|complete'"
```

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.15.33   True        True          10m     Working towards 4.16.14: 132 of 903 done
(14% complete), waiting on kube-controller-manager, kube-scheduler

NAME                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
MESSAGE
authentication             4.15.33   True        False         False      5d3h
baremetal                  4.15.33   True        False         False      5d4h
cloud-controller-manager   4.15.33   True        False         False      5d4h
cloud-credential           4.15.33   True        False         False      5d4h
cluster-autoscaler         4.15.33   True        False         False      5d4h
console                    4.15.33   True        False         False      5d3h
...
config-operator            4.16.14   True        False         False      5d4h
etcd                       4.16.14   True        False         False      5d4h
kube-apiserver             4.16.14   True        True          False      5d4h
NodeInstallerProgressing: 1 node is at revision 15; 2 nodes are at revision 17

NAME          STATUS   ROLES                AGE    VERSION
ctrl-plane-0  Ready    control-plane,master  5d4h  v1.28.13+2ca1a23
```

```
ctrl-plane-1   Ready    control-plane,master   5d4h   v1.28.13+2ca1a23
ctrl-plane-2   Ready    control-plane,master   5d4h   v1.28.13+2ca1a23
worker-0       Ready    mcp-1,worker           5d4h   v1.27.15+6147456
worker-1       Ready    mcp-2,worker           5d4h   v1.27.15+6147456

NAMESPACE                                       NAME                                                READY
STATUS      RESTARTS      AGE
openshift-kube-apiserver                        kube-apiserver-ctrl-plane-0
0/5     Pending     0           <invalid>
```

最のコ後ントロールプレーンノードが完了すると、クラスターのバージョンが新しい EUS リリースにすぐに更新されます。以下に例を示します。

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.16.14   True        False         123m    Cluster version is 4.16.14

NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
SINCE   MESSAGE
authentication                             4.16.14   True        False         False   5d6h
baremetal                                  4.16.14   True        False         False   5d7h
cloud-controller-manager                   4.16.14   True        False         False   5d7h
cloud-credential                           4.16.14   True        False         False   5d7h
cluster-autoscaler                         4.16.14   True        False         False   5d7h
config-operator                            4.16.14   True        False         False   5d7h
console                                    4.16.14   True        False         False   5d6h
#...
operator-lifecycle-manager-packageserver   4.16.14   True        False         False   5d7h
service-ca                                 4.16.14   True        False         False   5d7h
storage                                    4.16.14   True        False         False   5d7h

NAME          STATUS   ROLES                  AGE     VERSION
ctrl-plane-0   Ready    control-plane,master   5d7h    v1.29.8+f10c92d
ctrl-plane-1   Ready    control-plane,master   5d7h    v1.29.8+f10c92d
ctrl-plane-2   Ready    control-plane,master   5d7h    v1.29.8+f10c92d
worker-0       Ready    mcp-1,worker           5d7h    v1.27.15+6147456
worker-1       Ready    mcp-2,worker           5d7h    v1.27.15+6147456
```

関連情報

- クラスターの更新の監視

16.1.6.7. すべての OLM Operator の更新

複数バージョンのアップグレードの第 2 フェーズとして、すべての Operator を承認し、アップグレードするその他の Operator のインストールプランをさらに追加する必要があります。

「OLM Operator の更新」で説明されているのと同じ手順に従います。必要に応じて、OLM 以外の Operator も更新してください。

手順

1. クラスターの更新を監視します。たとえば、バージョン 4.14 から 4.15 へのクラスターの更新を監視するには、次のコマンドを実行します。

```
$ watch "oc get clusterversion; echo; oc get co | head -1; oc get co | grep 4.14; oc get co |
```

```
grep 4.15; echo; oc get no; echo; oc get po -A | grep -E -iv 'running|complete'"
```

2. 更新する必要がある Operator を確認します。

```
$ oc get installplan -A | grep -E 'APPROVED|false'
```

3. これらの Operator の **InstallPlan** リソースにパッチを適用します。

```
$ oc patch installplan -n metallb-system install-nwjnh --type merge --patch \
'{"spec":{"approved":true}}'
```

4. 次のコマンドを実行して namespace を監視します。

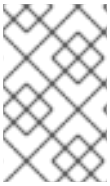
```
$ oc get all -n metallb-system
```

更新が完了すると、必要な Pod が **Running** 状態になり、必要な **ReplicaSet** リソースが準備完了になるはずです。

検証

更新中、この **watch** コマンドは、一度に 1 つまたは複数のクラスター Operator を循環し、**MESSAGE** 列に Operator 更新のステータスを表示します。

クラスター Operator の更新プロセスが完了すると、各コントロールプレーンノードが 1 つずつ再起動します。



注記

更新のこの部分で、クラスター Operator が再度更新中であることやデグレード状態であることを示すメッセージが報告されます。これは、ノードを再起動している間、コントロールプレーンノードがオフラインになるためです。

関連情報

- クラスターの更新の監視
- OLM Operator の更新

16.1.6.8. ワーカーノードの更新

コントロールプレーンを更新した後、作成した適切な **mcp** グループの一時停止を解除してワーカーノードをアップグレードします。**mcp** グループの一時停止を解除すると、そのグループ内のワーカーノードのアップグレードプロセスが開始します。クラスター内の各ワーカーノードが再起動し、必要に応じて新しい EUS、y-stream、または z-stream バージョンにアップグレードされます。

コントロールプレーンのみのアップグレードの場合、ワーカーノードの更新時に、再起動が 1 回だけ必要になり、ワーカーノードが <y+2> リリースバージョンにジャンプします。これは、大規模なベアメタルクラスターのアップグレードにかかる時間を短縮するために追加された機能です。

重要

この時点で保留することも可能です。ワーカーノードが <y-2> リリースであっても、新しい EUS リリースに更新したコントロールプレーンを使用して、実稼働環境での実行が完全にサポートされているクラスターバージョンを提供できます。これにより、大規模なクラスターを複数のメンテナンス期間にわたって段階的にアップグレードできます。

1. **mcp** グループ内で管理されているノードの数を確認できます。次のコマンドを実行して、**mcp** グループのリストを取得します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-c9a52144456dbff9c9af9c5a37d1b614   True      False      False
3            3                 3                   0                      36d
mcp-1    rendered-mcp-1-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                 0                   0                      47h
mcp-2    rendered-mcp-2-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                 0                   0                      47h
worker   rendered-worker-f1ab7b9a768e1b0ac9290a18817f60f0   True      False      False
0            0                 0                   0                      36d
```

注記

一度にアップグレードする **mcp** グループの数を決定してください。これは、一度に停止できる CNF Pod の数、および Pod Disruption Budget とアンチアフィニティーの設定よって異なります。

2. クラスター内のノードのリストを取得します。

```
$ oc get nodes
```

出力例

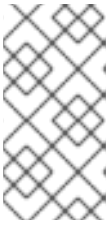
```
NAME          STATUS   ROLES                AGE    VERSION
ctrl-plane-0  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
worker-0      Ready    mcp-1,worker           5d8h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker           5d8h   v1.27.15+6147456
```

3. 一時停止中の **MachineConfigPool** グループを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   true
mcp-2   true
```



注記

MachineConfigPool は、それぞれ個別に一時停止解除できます。したがって、メンテナンス期間の残り時間がなくなった場合でも、他の MCP をすぐに一時停止解除する必要はありません。クラスターは、一部のワーカーノードをまだ <y-2> リリースバージョンのまま実行できます。

4. 必要な **mcp** グループを一時停止解除して、アップグレードを開始します。

```
$ oc patch mcp/mcp-1 --type merge --patch '{"spec":{"paused":false}}'
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/mcp-1 patched
```

5. 必要な **mcp** グループが一時停止解除されていることを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   true
```

6. 各 **mcp** グループがアップグレードされたら、残りのノードを一時停止解除してアップグレードを続けます。

```
$ oc get nodes
```

出力例

```
NAME          STATUS                        ROLES                AGE    VERSION
ctrl-plane-0  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
worker-0      Ready                         mcp-1,worker          5d8h   v1.29.8+f10c92d
worker-1      NotReady,SchedulingDisabled   mcp-2,worker          5d8h   v1.27.15+6147456
```

16.1.6.9. 新しく更新されたクラスターの健全性の確認

クラスターを更新した後、次のコマンドを実行して、クラスターが正常に稼働していることを確認します。

クラスターを更新した後、次のコマンドを実行して、クラスターが再び稼働していることを確認します。

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確認します。

```
$ oc get clusterversion
```

出力例

```
NAME     VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.16.14   True        False         4h38m   Cluster version is 4.16.14
```

新しいクラスターバージョンが返され、**PROGRESSING** 列に **False** が返されるはずです。

2. すべてのノードが準備完了であることを確認します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                AGE    VERSION
ctrl-plane-0   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-1   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-2   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
worker-0       Ready    mcp-1,worker           5d9h   v1.29.8+f10c92d
worker-1       Ready    mcp-2,worker           5d9h   v1.29.8+f10c92d
```

クラスター内のすべてのノードが **Ready** ステータスで、同じバージョンを実行しているはずです。

3. クラスター内に一時停止中の **mcp** リソースがないことを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   false
```

4. すべてのクラスター Operator が利用可能であることを確認します。

```
$ oc get co
```

出力例

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
```

```
SINCE   MESSAGE
authentication                             4.16.14   True        False         False      5d9h
baremetal                                  4.16.14   True        False         False      5d9h
cloud-controller-manager                   4.16.14   True        False         False      5d10h
cloud-credential                           4.16.14   True        False         False      5d10h
cluster-autoscaler                         4.16.14   True        False         False      5d9h
config-operator                            4.16.14   True        False         False      5d9h
console                                    4.16.14   True        False         False      5d9h
control-plane-machine-set                  4.16.14   True        False         False      5d9h
csi-snapshot-controller                    4.16.14   True        False         False      5d9h
dns                                        4.16.14   True        False         False      5d9h
etcd                                       4.16.14   True        False         False      5d9h
image-registry                             4.16.14   True        False         False      85m
ingress                                    4.16.14   True        False         False      5d9h
insights                                   4.16.14   True        False         False      5d9h
kube-apiserver                             4.16.14   True        False         False      5d9h
kube-controller-manager                    4.16.14   True        False         False      5d9h
kube-scheduler                             4.16.14   True        False         False      5d9h
kube-storage-version-migrator              4.16.14   True        False         False      4h48m
machine-api                                4.16.14   True        False         False      5d9h
machine-approver                           4.16.14   True        False         False      5d9h
machine-config                             4.16.14   True        False         False      5d9h
marketplace                                4.16.14   True        False         False      5d9h
monitoring                                 4.16.14   True        False         False      5d9h
network                                    4.16.14   True        False         False      5d9h
node-tuning                                4.16.14   True        False         False      5d7h
openshift-apiserver                        4.16.14   True        False         False      5d9h
openshift-controller-manager               4.16.14   True        False         False      5d9h
openshift-samples                          4.16.14   True        False         False      5h24m
operator-lifecycle-manager                 4.16.14   True        False         False      5d9h
operator-lifecycle-manager-catalog         4.16.14   True        False         False      5d9h
operator-lifecycle-manager-packageserver   4.16.14   True        False         False      5d9h
service-ca                                 4.16.14   True        False         False      5d9h
storage                                    4.16.14   True        False         False      5d9h
```

すべてのクラスター Operator の **AVAILABLE** 列に **True** と報告されるはずです。

5. すべての Pod が正常であることを確認します。

```
$ oc get po -A | grep -E -iv 'complete|running'
```

Pod は 1 つも返されないはずです。

注記

更新後もまだ移行中の Pod がいくつか見られる場合があります。しばらくこれを監視して、すべての Pod がクリアされることを確認してください。

16.1.7. y-stream クラスターの更新の完了

次の手順に従って、y-stream クラスターの更新を実行し、更新が完了するまで監視します。y-stream 更新の完了は、コントロールプレーンのみの更新よりも簡単です。

16.1.7.1. コントロールプレーンのみまたは y-stream の更新の承認

4.11 以降のすべてのバージョンに更新する場合は、更新の続行を手動で承認する必要があります。

重要

更新を承認する前に、更新先のバージョンで削除されている Kubernetes API を使用していないことを確認してください。たとえば、OpenShift Container Platform 4.17 では、API の削除はありません。詳細は、「Kubernetes API の削除」を参照してください。

前提条件

- クラスターで実行されているすべてのアプリケーションの API が、OpenShift Container Platform の次期 y-stream リリースと互換性があることを確認した。互換性の詳細は、「更新バージョン間のクラスター API バージョンの検証」を参照してください。

手順

- 次のコマンドを実行して、管理者としての承認を完了し、クラスターの更新を開始します。

```
$ oc adm upgrade
```

クラスターの更新が正常に完了しない場合は、**Reason** および **Message** セクションに更新の失敗に関する詳細が表示されます。

出力例

```
Cluster version is 4.15.45

Upgradeable=False

  Reason: MultipleReasons
  Message: Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"

ReleaseAccepted=False

  Reason: PreconditionChecks
  Message: Preconditions failed for payload loaded version="4.16.34" image="quay.io/openshift-release-dev/ocp-release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d520b353b8": Precondition "ClusterVersionUpgradeable" failed because of "MultipleReasons": Cluster should not be upgraded between minor versions for multiple reasons: AdminAckRequired,ResourceDeletesInProgress
  * Kubernetes 1.29 and therefore OpenShift 4.16 remove several APIs which require admin consideration. Please see the knowledge article https://access.redhat.com/articles/7031404 for details and instructions.
  * Cluster minor level upgrades are not allowed while resource deletions are in progress; resources=PrometheusRule "openshift-kube-apiserver/kube-apiserver-recording-rules"

Upstream is unset, so the cluster will use an appropriate default.
Channel: eus-4.16 (available channels: candidate-4.15, candidate-4.16, eus-4.16, fast-4.15,
```

```
fast-4.16, stable-4.15, stable-4.16)

Recommended updates:

  VERSION     IMAGE
  4.16.34     quay.io/openshift-release-dev/ocp-release@sha256:41bb08c560f6db5039ccdf242e590e8b23049b5eb31e1c4f6021d1d1d520b353b8
```

注記

この例は、Red Hat のナレッジベース記事 (Preparing to upgrade to OpenShift Container Platform 4.16) に、リリース間の API 互換性の検証に関する詳細が記載されています。

検証

- 次のコマンドを実行して更新を確認します。

```
$ oc get configmap admin-acks -n openshift-config -o json | jq .data
```

出力例

```
{
  "ack-4.14-kube-1.28-api-removals-in-4.15": "true",
  "ack-4.15-kube-1.29-api-removals-in-4.16": "true"
}
```

注記

この例では、コントロールプレーンのみの更新で、クラスターをバージョン 4.14 から 4.15 に更新してから、4.15 から 4.16 に更新します。

関連情報

- Kubernetes API の削除
- 更新バージョン間のクラスター API バージョンの検証

16.1.7.2. クラスターの更新の開始

ある y-stream リリースから次のリリースに更新する場合、中間の z-stream リリースにも互換性があることを確認する必要があります。

注記

oc adm upgrade コマンドを実行すると、更新先のリリースが実行可能なリリースであることを確認できます。**oc adm upgrade** コマンドは、互換性のある更新リリースをリスト表示します。

手順

1. 更新を開始します。

```
$ oc adm upgrade --to=4.15.33
```

重要

- **コントロールプレーンのみの更新**: 中間の <y+1> リリースパスを指定します。
- **y-stream 更新** - Kubernetes バージョンスキューポリシー に準拠した正しい <y.z> リリースを使用していることを確認します。
- **z-stream の更新** - 特定のリリースへの移行に問題がないことを確認します。

出力例

```
Requested update to 4.15.33 1
```

1 **Requested update** 値は、実際の更新に応じて変わります。

関連情報

- ターゲットリリースの選択

16.1.7.3. クラスターの更新の監視

更新時には、クラスターの健全性を頻繁に確認する必要があります。ノードのステータス、クラスター Operator のステータス、失敗した Pod を確認します。

手順

- クラスターの更新を監視します。たとえば、バージョン 4.14 から 4.15 へのクラスターの更新を監視するには、次のコマンドを実行します。

```
$ watch "oc get clusterversion; echo; oc get co | head -1; oc get co | grep 4.14; oc get co | grep 4.15; echo; oc get no; echo; oc get po -A | grep -E -iv 'running|complete'"
```

出力例

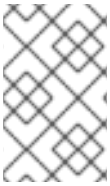
```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.14.34   True        True          4m6s    Working towards 4.15.33: 111 of 873 done (12% complete), waiting on kube-apiserver

NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
authentication                             4.14.34   True        False         False      4d22h
baremetal                                  4.14.34   True        False         False      4d23h
cloud-controller-manager                   4.14.34   True        False         False      4d23h
cloud-credential                           4.14.34   True        False         False      4d23h
cluster-autoscaler                         4.14.34   True        False         False      4d23h
console                                    4.14.34   True        False         False      4d22h

...
```

```
storage                    4.14.34   True       False       False      4d23h
config-operator            4.15.33   True       False       False      4d23h
etcd                       4.15.33   True       False       False      4d23h

NAME          STATUS   ROLES                AGE     VERSION
ctrl-plane-0  Ready    control-plane,master   4d23h   v1.27.15+6147456
ctrl-plane-1  Ready    control-plane,master   4d23h   v1.27.15+6147456
ctrl-plane-2  Ready    control-plane,master   4d23h   v1.27.15+6147456
worker-0      Ready    mcp-1,worker           4d23h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker           4d23h   v1.27.15+6147456

NAMESPACE             NAME                    READY   STATUS              RESTARTS   AGE
openshift-marketplace   redhat-marketplace-rf86t   0/1     ContainerCreating   0          0s
```

検証

更新中、この **watch** コマンドは、一度に 1 つまたは複数のクラスター Operator を循環し、**MESSAGE** 列に Operator 更新のステータスを表示します。

クラスター Operator の更新プロセスが完了すると、各コントロールプレーンノードが 1 つずつ再起動します。



注記

更新のこの部分で、クラスター Operator が再度更新中であることやデグレード状態であることを示すメッセージが報告されます。これは、ノードを再起動している間、コントロールプレーンノードがオフラインになるためです。

最後のコントロールプレーンノードの再起動が完了するとすぐに、更新されたクラスターのバージョンが表示されます。

コントロールプレーンの更新が完了すると、次のようなメッセージが表示されます。この例では、中間の y-stream リリースへの更新が完了したことを示しています。

```
NAME     VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.15.33   True        False         28m     Cluster version is 4.15.33

NAME                       VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
authentication             4.15.33   True        False         False      5d
baremetal                  4.15.33   True        False         False      5d
cloud-controller-manager   4.15.33   True        False         False      5d1h
cloud-credential           4.15.33   True        False         False      5d1h
cluster-autoscaler         4.15.33   True        False         False      5d
config-operator            4.15.33   True        False         False      5d
console                    4.15.33   True        False         False      5d

...

service-ca                 4.15.33   True        False         False      5d
storage                    4.15.33   True        False         False      5d

NAME          STATUS   ROLES                AGE   VERSION
ctrl-plane-0  Ready    control-plane,master   5d    v1.28.13+2ca1a23
ctrl-plane-1  Ready    control-plane,master   5d    v1.28.13+2ca1a23
```

```
ctrl-plane-2   Ready   control-plane,master   5d   v1.28.13+2ca1a23
worker-0       Ready   mcp-1,worker          5d   v1.28.13+2ca1a23
worker-1       Ready   mcp-2,worker          5d   v1.28.13+2ca1a23
```

16.1.7.4. OLM Operator の更新

通信事業者の環境では、ソフトウェアを実稼働クラスターにロードする前に検査する必要があります。また、実稼働クラスターが非接続ネットワーク内で設定されるため、必ずしもインターネットに直接接続されているわけではありません。クラスターが非接続ネットワーク内にあるため、新しいバージョンをクラスターごとに管理できるように、インストール時に OpenShift Operator を手動で更新するように設定します。Operator を新しいバージョンに移行するには、次の手順を実行します。

手順

1. 更新する必要がある Operator を確認します。

```
$ oc get installplan -A | grep -E 'APPROVED|false'
```

出力例

```
NAMESPACE          NAME           CSV                                        APPROVAL
APPROVED
metallb-system     install-nwjnh  metallb-operator.v4.16.0-202409202304      Manual
false
openshift-nmstate  install-5r7wr  kubernetes-nmstate-operator.4.16.0-202409251605
Manual     false
```

2. これらの Operator の **InstallPlan** リソースにパッチを適用します。

```
$ oc patch installplan -n metallb-system install-nwjnh --type merge --patch \
'{"spec":{"approved":true}}'
```

出力例

```
installplan.operators.coreos.com/install-nwjnh patched
```

3. 次のコマンドを実行して namespace を監視します。

```
$ oc get all -n metallb-system
```

出力例

```
NAME                                                READY   STATUS              RESTARTS   AGE
pod/metallb-operator-controller-manager-69b5f884c-8bp22    0/1     ContainerCreating   0
4s
pod/metallb-operator-controller-manager-77895bdb46-bqjdx   1/1     Running             0
4m1s
pod/metallb-operator-webhook-server-5d9b968896-vnbhk       0/1     ContainerCreating   0
4s
pod/metallb-operator-webhook-server-d76f9c6c8-57r4w        1/1     Running             0
4m1s

...
```

```
NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/metallb-operator-controller-manager-69b5f884c    1         1         0       4s
replicaset.apps/metallb-operator-controller-manager-77895bdb46   1         1         1       4m1s
replicaset.apps/metallb-operator-controller-manager-99b76f88     0         0         0       4m40s
replicaset.apps/metallb-operator-webhook-server-5d9b968896       1         1         0       4s
replicaset.apps/metallb-operator-webhook-server-6f7dbfdb88       0         0         0       4m40s
replicaset.apps/metallb-operator-webhook-server-d76f9c6c8        1         1         1       4m1s
```

更新が完了すると、必要な Pod が **Running** 状態になり、必要な **ReplicaSet** リソースが準備完了になるはずです。

```
NAME                                              READY   STATUS    RESTARTS   AGE
pod/metallb-operator-controller-manager-69b5f884c-8bp22   1/1     Running   0          25s
pod/metallb-operator-webhook-server-5d9b968896-vnbhk      1/1     Running   0          25s

...

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/metallb-operator-controller-manager-69b5f884c    1         1         1       25s
replicaset.apps/metallb-operator-controller-manager-77895bdb46   0         0         0       4m22s
replicaset.apps/metallb-operator-webhook-server-5d9b968896       1         1         1       25s
replicaset.apps/metallb-operator-webhook-server-d76f9c6c8        0         0         0       4m22s
```

検証

- Operator を再度更新する必要がないことを確認します。

```
$ oc get installplan -A | grep -E 'APPROVED|false'
```

出力は返されないはずです。

注記

Operator によっては、最終バージョンの前にインストールする必要がある中間の z-stream リリースバージョンがあるため、更新を 2 回承認する必要がある場合があります。

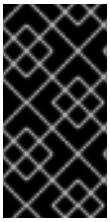
関連情報

- ワーカーノードの更新

16.1.7.5. ワーカーノードの更新

コントロールプレーンを更新した後、作成した適切な **mcp** グループの一時停止を解除してワーカーノードをアップグレードします。**mcp** グループの一時停止を解除すると、そのグループ内のワーカーノードのアップグレードプロセスが開始します。クラスター内の各ワーカーノードが再起動し、必要に応じて新しい EUS、y-stream、または z-stream バージョンにアップグレードされます。

コントロールプレーンのみのアップグレードの場合、ワーカーノードの更新時に、再起動が 1 回だけ必要になり、ワーカーノードが <y+2> リリースバージョンにジャンプします。これは、大規模なベアメタルクラスターのアップグレードにかかる時間を短縮するために追加された機能です。



重要

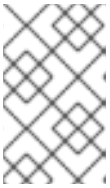
この時点で保留することも可能です。ワーカーノードが <y-2> リリースであっても、新しい EUS リリースに更新したコントロールプレーンを使用して、実稼働環境での実行が完全にサポートされているクラスターバージョンを提供できます。これにより、大規模なクラスターを複数のメンテナンス期間にわたって段階的にアップグレードできます。

1. **mcp** グループ内で管理されているノードの数を確認できます。次のコマンドを実行して、**mcp** グループのリストを取得します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-c9a52144456dbff9c9af9c5a37d1b614   True      False      False
3            3                 3                   0                      36d
mcp-1    rendered-mcp-1-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                 0                   0                      47h
mcp-2    rendered-mcp-2-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                 0                   0                      47h
worker   rendered-worker-f1ab7b9a768e1b0ac9290a18817f60f0   True      False      False
0            0                 0                   0                      36d
```



注記

一度にアップグレードする **mcp** グループの数を決定してください。これは、一度に停止できる CNF Pod の数、および Pod Disruption Budget とアンチアフィニティーの設定よって異なります。

2. クラスター内のノードのリストを取得します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                AGE    VERSION
ctrl-plane-0  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
worker-0      Ready    mcp-1,worker           5d8h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker           5d8h   v1.27.15+6147456
```

3. 一時停止中の **MachineConfigPool** グループを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   true
mcp-2   true
```

注記

MachineConfigPool は、それぞれ個別に一時停止解除できます。したがって、メンテナンス期間の残り時間がなくなった場合でも、他の MCP をすぐに一時停止解除する必要はありません。クラスターは、一部のワーカーノードをまだ <y-2> リリースバージョンのまま実行できます。

4. 必要な **mcp** グループを一時停止解除して、アップグレードを開始します。

```
$ oc patch mcp/mcp-1 --type merge --patch '{"spec":{"paused":false}}'
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/mcp-1 patched
```

5. 必要な **mcp** グループが一時停止解除されていることを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   true
```

6. 各 **mcp** グループがアップグレードされたら、残りのノードを一時停止解除してアップグレードを続けます。

```
$ oc get nodes
```

出力例

```
NAME          STATUS                        ROLES                AGE    VERSION
ctrl-plane-0  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready                         control-plane,master  5d8h   v1.29.8+f10c92d
worker-0      Ready                         mcp-1,worker          5d8h   v1.29.8+f10c92d
worker-1      NotReady,SchedulingDisabled   mcp-2,worker          5d8h   v1.27.15+6147456
```

16.1.7.6. 新しく更新されたクラスターの健全性の確認

クラスターを更新した後、次のコマンドを実行して、クラスターが再び稼働していることを確認します。

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確認します。

```
$ oc get clusterversion
```

出力例

```
NAME     VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.16.14   True        False         4h38m   Cluster version is 4.16.14
```

新しいクラスターバージョンが返され、**PROGRESSING** 列に **False** が返されるはずです。

2. すべてのノードが準備完了であることを確認します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                AGE    VERSION
ctrl-plane-0   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-1   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-2   Ready    control-plane,master   5d9h   v1.29.8+f10c92d
worker-0       Ready    mcp-1,worker           5d9h   v1.29.8+f10c92d
worker-1       Ready    mcp-2,worker           5d9h   v1.29.8+f10c92d
```

クラスター内のすべてのノードが **Ready** ステータスで、同じバージョンを実行しているはずです。

3. クラスター内に一時停止中の **mcp** リソースがないことを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   false
```

4. すべてのクラスター Operator が利用可能であることを確認します。

```
$ oc get co
```

出力例

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
```

```
SINCE   MESSAGE
authentication                             4.16.14   True        False         False      5d9h
baremetal                                  4.16.14   True        False         False      5d9h
cloud-controller-manager                   4.16.14   True        False         False      5d10h
cloud-credential                           4.16.14   True        False         False      5d10h
cluster-autoscaler                         4.16.14   True        False         False      5d9h
config-operator                            4.16.14   True        False         False      5d9h
console                                    4.16.14   True        False         False      5d9h
control-plane-machine-set                  4.16.14   True        False         False      5d9h
csi-snapshot-controller                    4.16.14   True        False         False      5d9h
dns                                        4.16.14   True        False         False      5d9h
etcd                                       4.16.14   True        False         False      5d9h
image-registry                             4.16.14   True        False         False      85m
ingress                                    4.16.14   True        False         False      5d9h
insights                                   4.16.14   True        False         False      5d9h
kube-apiserver                             4.16.14   True        False         False      5d9h
kube-controller-manager                    4.16.14   True        False         False      5d9h
kube-scheduler                             4.16.14   True        False         False      5d9h
kube-storage-version-migrator              4.16.14   True        False         False      4h48m
machine-api                                4.16.14   True        False         False      5d9h
machine-approver                           4.16.14   True        False         False      5d9h
machine-config                             4.16.14   True        False         False      5d9h
marketplace                                4.16.14   True        False         False      5d9h
monitoring                                 4.16.14   True        False         False      5d9h
network                                    4.16.14   True        False         False      5d9h
node-tuning                                4.16.14   True        False         False      5d7h
openshift-apiserver                        4.16.14   True        False         False      5d9h
openshift-controller-manager               4.16.14   True        False         False      5d9h
openshift-samples                          4.16.14   True        False         False      5h24m
operator-lifecycle-manager                 4.16.14   True        False         False      5d9h
operator-lifecycle-manager-catalog         4.16.14   True        False         False      5d9h
operator-lifecycle-manager-packageserver   4.16.14   True        False         False      5d9h
service-ca                                 4.16.14   True        False         False      5d9h
storage                                    4.16.14   True        False         False      5d9h
```

すべてのクラスター Operator の **AVAILABLE** 列に **True** と報告されるはずです。

5. すべての Pod が正常であることを確認します。

```
$ oc get po -A | grep -E -iv 'complete|running'
```

Pod は 1 つも返されないはずです。

注記

更新後もまだ移行中の Pod がいくつか見られる場合があります。しばらくこれを監視して、すべての Pod がクリアされることを確認してください。

16.1.8. z-stream クラスターの更新の完了

次の手順に従って、z-stream クラスターの更新を実行し、更新が完了するまで監視します。z-stream 更新の完了は、コントロールプレーンのみの更新や y-stream 更新よりも簡単です。

16.1.8.1. クラスターの更新の開始

ある y-stream リリースから次のリリースに更新する場合、中間の z-stream リリースにも互換性があることを確認する必要があります。

注記

oc adm upgrade コマンドを実行すると、更新先のリリースが実行可能なリリースであることを確認できます。**oc adm upgrade** コマンドは、互換性のある更新リリースをリスト表示します。

手順

1. 更新を開始します。

```
$ oc adm upgrade --to=4.15.33
```

重要

- **コントロールプレーンのみの更新**: 中間の <y+1> リリースパスを指定します。
- **y-stream 更新** - Kubernetes バージョンスキューポリシー に準拠した正しい <y.z> リリースを使用していることを確認します。
- **z-stream の更新** - 特定のリリースへの移行に問題がないことを確認します。

出力例

```
Requested update to 4.15.33 1
```

1 **Requested update** 値は、実際の更新に応じて変わります。

関連情報

- ターゲットリリースの選択

16.1.8.2. ワーカーノードの更新

コントロールプレーンを更新した後、作成した適切な **mcp** グループの一時停止を解除してワーカーノードをアップグレードします。**mcp** グループの一時停止を解除すると、そのグループ内のワーカーノードのアップグレードプロセスが開始します。クラスター内の各ワーカーノードが再起動し、必要に応じて新しい EUS、y-stream、または z-stream バージョンにアップグレードされます。

コントロールプレーンのみのアップグレードの場合、ワーカーノードの更新時に、再起動が 1 回だけ必要になり、ワーカーノードが <y+2> リリースバージョンにジャンプします。これは、大規模なベアメタルクラスターのアップグレードにかかる時間を短縮するために追加された機能です。

重要

この時点で保留することも可能です。ワーカーノードが <y-2> リリースであっても、新しい EUS リリースに更新したコントロールプレーンを使用して、実稼働環境での実行が完全にサポートされているクラスターバージョンを提供できます。これにより、大規模なクラスターを複数のメンテナンス期間にわたって段階的にアップグレードできます。

1. **mcp** グループ内で管理されているノードの数を確認できます。次のコマンドを実行して、**mcp** グループのリストを取得します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-c9a52144456dbff9c9af9c5a37d1b614   True      False      False
3            3                3                   0                   36d
mcp-1    rendered-mcp-1-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                0                   0                   47h
mcp-2    rendered-mcp-2-07fe50b9ad51fae43ed212e84e1dcc8e    False     False      False
1            0                0                   0                   47h
worker   rendered-worker-f1ab7b9a768e1b0ac9290a18817f60f0   True      False      False
0            0                0                   0                   36d
```

注記

一度にアップグレードする **mcp** グループの数を決定してください。これは、一度に停止できる CNF Pod の数、および Pod Disruption Budget とアンチアフィニティーの設定よって異なります。

2. クラスター内のノードのリストを取得します。

```
$ oc get nodes
```

出力例

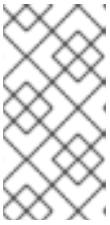
```
NAME          STATUS   ROLES                  AGE    VERSION
ctrl-plane-0  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready    control-plane,master   5d8h   v1.29.8+f10c92d
worker-0      Ready    mcp-1,worker           5d8h   v1.27.15+6147456
worker-1      Ready    mcp-2,worker           5d8h   v1.27.15+6147456
```

3. 一時停止中の **MachineConfigPool** グループを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name),
(.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   true
mcp-2   true
```



注記

MachineConfigPool は、それぞれ個別に一時停止解除できます。したがって、メンテナンス期間の残り時間がなくなった場合でも、他の MCP をすぐに一時停止解除する必要はありません。クラスターは、一部のワーカーノードをまだ <y-2> リリースバージョンのまま実行できます。

4. 必要な **mcp** グループを一時停止解除して、アップグレードを開始します。

```
$ oc patch mcp/mcp-1 --type merge --patch '{"spec":{"paused":false}}'
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/mcp-1 patched
```

5. 必要な **mcp** グループが一時停止解除されていることを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   true
```

6. 各 **mcp** グループがアップグレードされたら、残りのノードを一時停止解除してアップグレードを続けます。

```
$ oc get nodes
```

出力例

```
NAME          STATUS                     ROLES                AGE    VERSION
ctrl-plane-0  Ready                      control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-1  Ready                      control-plane,master  5d8h   v1.29.8+f10c92d
ctrl-plane-2  Ready                      control-plane,master  5d8h   v1.29.8+f10c92d
worker-0      Ready                      mcp-1,worker          5d8h   v1.29.8+f10c92d
worker-1      NotReady,SchedulingDisabled  mcp-2,worker          5d8h   v1.27.15+6147456
```

16.1.8.3. 新しく更新されたクラスターの健全性の確認

クラスターを更新した後、次のコマンドを実行して、クラスターが再び稼働していることを確認します。

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確認します。

```
$ oc get clusterversion
```

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.16.14   True        False         4h38m   Cluster version is 4.16.14
```

新しいクラスターバージョンが返され、**PROGRESSING** 列に **False** が返されるはずです。

2. すべてのノードが準備完了であることを確認します。

```
$ oc get nodes
```

出力例

```
NAME          STATUS   ROLES                  AGE    VERSION
ctrl-plane-0  Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-1  Ready    control-plane,master   5d9h   v1.29.8+f10c92d
ctrl-plane-2  Ready    control-plane,master   5d9h   v1.29.8+f10c92d
worker-0      Ready    mcp-1,worker           5d9h   v1.29.8+f10c92d
worker-1      Ready    mcp-2,worker           5d9h   v1.29.8+f10c92d
```

クラスター内のすべてのノードが **Ready** ステータスで、同じバージョンを実行しているはずです。

3. クラスター内に一時停止中の **mcp** リソースがないことを確認します。

```
$ oc get mcp -o json | jq -r '["MCP","Paused"], ["---","------"], (.items[] | [(.metadata.name), (.spec.paused)]) | @tsv' | grep -v worker
```

出力例

```
MCP     Paused
---     ------
master  false
mcp-1   false
mcp-2   false
```

4. すべてのクラスター Operator が利用可能であることを確認します。

```
$ oc get co
```

出力例

```
NAME                                       VERSION   AVAILABLE   PROGRESSING   DEGRADED
SINCE   MESSAGE
authentication                             4.16.14   True        False         False      5d9h
baremetal                                  4.16.14   True        False         False      5d9h
cloud-controller-manager                   4.16.14   True        False         False      5d10h
cloud-credential                           4.16.14   True        False         False      5d10h
cluster-autoscaler                         4.16.14   True        False         False      5d9h
config-operator                            4.16.14   True        False         False      5d9h
console                                    4.16.14   True        False         False      5d9h
control-plane-machine-set                  4.16.14   True        False         False      5d9h
```

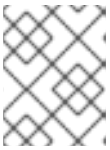
```
csi-snapshot-controller                 4.16.14   True        False         False      5d9h
dns                                     4.16.14   True        False         False      5d9h
etcd                                    4.16.14   True        False         False      5d9h
image-registry                          4.16.14   True        False         False      85m
ingress                                 4.16.14   True        False         False      5d9h
insights                                4.16.14   True        False         False      5d9h
kube-apiserver                          4.16.14   True        False         False      5d9h
kube-controller-manager                 4.16.14   True        False         False      5d9h
kube-scheduler                          4.16.14   True        False         False      5d9h
kube-storage-version-migrator           4.16.14   True        False         False      4h48m
machine-api                             4.16.14   True        False         False      5d9h
machine-approver                        4.16.14   True        False         False      5d9h
machine-config                          4.16.14   True        False         False      5d9h
marketplace                             4.16.14   True        False         False      5d9h
monitoring                              4.16.14   True        False         False      5d9h
network                                 4.16.14   True        False         False      5d9h
node-tuning                             4.16.14   True        False         False      5d7h
openshift-apiserver                     4.16.14   True        False         False      5d9h
openshift-controller-manager            4.16.14   True        False         False      5d9h
openshift-samples                       4.16.14   True        False         False      5h24m
operator-lifecycle-manager              4.16.14   True        False         False      5d9h
operator-lifecycle-manager-catalog      4.16.14   True        False         False      5d9h
operator-lifecycle-manager-packageserver 4.16.14  True        False         False      5d9h
service-ca                              4.16.14   True        False         False      5d9h
storage                                 4.16.14   True        False         False      5d9h
```

すべてのクラスター Operator の **AVAILABLE** 列に **True** と報告されるはずです。

5. すべての Pod が正常であることを確認します。

```
$ oc get po -A | grep -E -iv 'complete|running'
```

Pod は 1 つも返されないはずです。



注記

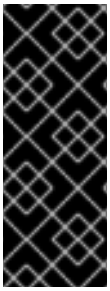
更新後もまだ移行中の Pod がいくつか見られる場合があります。しばらくこれを監視して、すべての Pod がクリアされることを確認してください。

16.2. 通信事業者コア CNF クラスターのトラブルシューティングとメンテナンス

16.2.1. 通信事業者コア CNF クラスターのトラブルシューティングとメンテナンス

トラブルシューティングとメンテナンスは週次のタスクであり、コンポーネントを更新する場合でも、問題を調査する場合でも、目標を達成するためのツールがなければ困難になる可能性があります。ツールや回答をどこでどのように検索するかを知ることも、課題の 1 つです。

高帯域幅のネットワークスループットが必要なベアメタル環境をメンテナンスおよびトラブルシューティングするには、次の手順を参照してください。



重要

このトラブルシューティング情報は、OpenShift Container Platform の設定やクラウドネイティブネットワーク機能 (CNF) アプリケーションの開発に関する参考資料ではありません。

通信事業者の CNF アプリケーションの開発については、Red Hat Best Practices for Kubernetes を参照してください。

16.2.1.1. クラウドネイティブネットワーク機能

通信事業者のクラウドネイティブネットワーク機能 (CNF) アプリケーションに OpenShift Container Platform の使用を開始する場合は、CNF を理解すると、発生する可能性のある問題を把握するのに役立ちます。

CNF とその進化の詳細は、VNF and CNF, what's the difference? を参照してください。

16.2.1.2. サポートの利用

手順で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。カスタマーポータルから、次のようにさまざまな方法でヘルプを見つけることができます。

- Red Hat 製品に関する記事やソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

デプロイメントの問題を特定するには、デバッグツールを使用するか、デプロイメントのヘルスエンドポイントを確認します。デプロイメントのデバッグまたは健全性情報の取得が完了したら、Red Hat ナレッジベースでソリューションを検索したり、サポートチケットを提出したりできます。

16.2.1.2.1. Red Hat ナレッジベースについて

Red Hat ナレッジベース は、お客様が Red Hat の製品やテクノロジーを最大限に活用できるようにするための豊富なコンテンツを提供します。Red Hat ナレッジベースは、Red Hat 製品のインストール、設定、および使用に関する記事、製品ドキュメント、および動画で構成されています。さらに、既知の問題に対する解決策を検索でき、それぞれに根本原因の簡潔な説明と修復手順が記載されています。

16.2.1.2.2. Red Hat ナレッジベースの検索

OpenShift Container Platform の問題が発生した場合には、初期検索を実行して、解決策を Red Hat ナレッジベース内ですでに見つけることができるかどうかを確認できます。

前提条件

- Red Hat カスタマーポータルのアカウントがある。

手順

1. Red Hat カスタマーポータル にログインします。
2. **Search** をクリックします。

3. 検索フィールドに、問題に関連する次のようなキーワードと文字列を入力します。
   - OpenShift Container Platform コンポーネント (**etcd** など)
   - 関連する手順 (**installation** など)
   - 明示的な失敗に関連する警告、エラーメッセージ、およびその他の出力
4. **Enter** キーをクリックします。
5. オプション: **OpenShift Container Platform** 製品フィルターを選択します。
6. オプション: **Documentation** コンテンツタイプフィルターを選択します。

16.2.1.2.3. サポートケースの作成

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Red Hat カスタマーポータルのアカウントがある。
- Red Hat の Standard または Premium サブスクリプションがある。

手順

1. Red Hat カスタマーポータルの Customer Support ページ にログインします。
2. **Get support** をクリックします。
3. **Customer Support** ページの **Cases** タブで、以下を行います。
   a. オプション: 必要に応じて、事前に入力されたアカウントと所有者の詳細を変更します。
   b. 問題に該当するカテゴリー (**Bug**、**Defect** など) を選択し、**Continue** をクリックします。
4. 以下の情報を入力します。
   a. **Summary** フィールドには、問題の簡潔で説明的な概要と、確認されている現象および予想される動作の詳細情報を入力します。
   b. **Product** ドロップダウンメニューから **OpenShift Container Platform** を選択します。
   c. **Version** ドロップダウンから **4.16** を選択します。
5. Red Hat ナレッジベースで推奨されるソリューション一覧を確認してください。この一覧に上げられているソリューションは、報告しようとしている問題に適用される可能性があります。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。
6. 報告している問題に対する一致に基づいて推奨される Red Hat ナレッジベースソリューションの一覧が更新されることを確認してください。ケース作成プロセスでより多くの情報を提供すると、このリストの絞り込みが行われます。提案されている記事が問題に対応していない場合は、**Continue** をクリックします。

7. アカウント情報が予想通りに表示されていることを確認し、そうでない場合は適宜修正します。
8. 自動入力された OpenShift Container Platform クラスター ID が正しいことを確認します。正しくない場合は、クラスター ID を手動で取得します。
   - OpenShift Container Platform Web コンソールを使用してクラスター ID を手動で取得するには、以下を実行します。
     a. **Home → Overview** に移動します。
     b. **Details** セクションの **Cluster ID** フィールドで値を見つけます。
   - または、OpenShift Container Platform Web コンソールで新規サポートケースを作成し、クラスター ID を自動的に入力することができます。
     a. ツールバーから、**(?)Help → Open Support Case** に移動します。
     b. **Cluster ID** 値が自動的に入力されます。
   - OpenShift CLI (**oc**) を使用してクラスター ID を取得するには、以下のコマンドを実行します。

     ```
     $ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
     ```
9. プロンプトが表示されたら、以下の質問に回答し、**Continue** をクリックします。
   - What are you experiencing? What are you expecting to happen?
   - Define the value or impact to you or the business.
   - Where are you experiencing this behavior? What environment?
   - When does this behavior occur? Frequency? Repeatedly? At certain times?
10. 関連する診断データファイルをアップロードし、**Continue** をクリックします。まずは、**oc adm must-gather** コマンドを使用して収集されるデータと、そのコマンドによって収集されない問題に固有のデータを含めることが推奨されます。
11. 関連するケース管理の詳細情報を入力し、**Continue** をクリックします。
12. ケースの詳細をプレビューし、**Submit** をクリックします。

16.2.2. 一般的なトラブルシューティング

問題が発生した場合は、まず問題が発生している特定の領域を確認します。問題が発生する可能性のある領域を絞り込むには、次の 1 つ以上のタスクを完了します。

- クラスターのクエリー
- Pod ログの確認
- Pod のデバッグ
- イベントの確認

16.2.2.1. クラスターのクエリー

潜在的な問題をより正確に発見できるように、クラスターに関する情報を取得します。

手順

1. 次のコマンドを実行してプロジェクトに切り替えます。

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

2. 次のコマンドを実行して、クラスターのバージョン、クラスター Operator、およびその namespace 内のノードをクエリーします。

```
$ oc get clusterversion,clusteroperator,node
```

出力例

```
NAME                                         VERSION   AVAILABLE   PROGRESSING   SINCE
STATUS
clusterversion.config.openshift.io/version   4.16.11   True        False         62d     Cluster
version is 4.16.11

NAME                                                                VERSION   AVAILABLE
PROGRESSING   DEGRADED   SINCE   MESSAGE
clusteroperator.config.openshift.io/authentication                  4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/baremetal                       4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/cloud-controller-manager        4.16.11   True
False         False      62d
clusteroperator.config.openshift.io/cloud-credential                4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/cluster-autoscaler              4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/config-operator                 4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/console                         4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/control-plane-machine-set       4.16.11   True
False         False      62d
clusteroperator.config.openshift.io/csi-snapshot-controller         4.16.11   True
False         False      62d
clusteroperator.config.openshift.io/dns                             4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/etcd                            4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/image-registry                  4.16.11   True        False
False      55d
clusteroperator.config.openshift.io/ingress                         4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/insights                        4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/kube-apiserver                  4.16.11   True        False
False      62d
clusteroperator.config.openshift.io/kube-controller-manager         4.16.11   True
False         False      62d
clusteroperator.config.openshift.io/kube-scheduler                  4.16.11   True        False
```

```
False     62d
clusteroperator.config.openshift.io/kube-storage-version-migrator            4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/machine-api                              4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/machine-approver                         4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/machine-config                           4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/marketplace                              4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/monitoring                               4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/network                                  4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/node-tuning                              4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/openshift-apiserver                      4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/openshift-controller-manager             4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/openshift-samples                        4.16.11   True
False       False     35d
clusteroperator.config.openshift.io/operator-lifecycle-manager               4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/operator-lifecycle-manager-catalog       4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/operator-lifecycle-manager-packageserver 4.16.11   True
False       False     62d
clusteroperator.config.openshift.io/service-ca                               4.16.11   True       False
False     62d
clusteroperator.config.openshift.io/storage                                  4.16.11   True       False
False     62d

NAME              STATUS   ROLES                       AGE   VERSION
node/ctrl-plane-0   Ready    control-plane,master,worker   62d   v1.29.7
node/ctrl-plane-1   Ready    control-plane,master,worker   62d   v1.29.7
node/ctrl-plane-2   Ready    control-plane,master,worker   62d   v1.29.7
```

詳細は、「oc get」および「Pod ステータスの確認」を参照してください。

関連情報

- oc get
- Pod ステータスの確認

16.2.2.2. Pod ログの確認

Pod からログを取得して、ログで問題を確認します。

手順

1. 次のコマンドを実行して、Pod を一覧表示します。

```
$ oc get pod
```

出力例

```
NAME       READY   STATUS    RESTARTS        AGE
busybox-1   1/1     Running   168 (34m ago)     7d
busybox-2   1/1     Running   119 (9m20s ago)   4d23h
busybox-3   1/1     Running   168 (43m ago)     7d
busybox-4   1/1     Running   168 (43m ago)     7d
```

2. 次のコマンドを実行して、Pod ログファイルを確認します。

```
$ oc logs -n <namespace> busybox-1
```

詳細は、「oc ログ」、「ロギング」、および「Pod およびコンテナーログの検査」を参照してください。

関連情報

- oc logs
- ロギング
- Pod およびコンテナーログの検査

16.2.2.3. Pod に対する describe の実行

Pod に対して describe を実行すると、その Pod に関する情報が得られ、トラブルシューティングに役立ちます。**Events** セクションに、Pod とその中のコンテナーに関する詳細情報が表示されます。

手順

- 次のコマンドを実行して Pod に関する情報を取得します。

```
$ oc describe pod -n <namespace> busybox-1
```

出力例

```
Name:             busybox-1
Namespace:        busy
Priority:         0
Service Account:  default
Node:             worker-3/192.168.0.0
Start Time:       Mon, 27 Nov 2023 14:41:25 -0500
Labels:           app=busybox
                  pod-template-hash=<hash>
Annotations:      k8s.ovn.org/pod-networks:
…
Events:
  Type    Reason   Age                From     Message
  ----    ------   ----               ----     -------
  Normal  Pulled   41m (x170 over 7d1h)  kubelet  Container image
```

```
"quay.io/quay/busybox:latest" already present on machine
  Normal  Created  41m (x170 over 7d1h)  kubelet  Created container busybox
  Normal  Started  41m (x170 over 7d1h)  kubelet  Started container busybox
```

詳細は、「oc describe」を参照してください。

関連情報

- oc describe

16.2.2.4. イベントの確認

特定の namespace 内のイベントを確認すると、潜在的な問題を発見できます。

手順

1. 次のコマンドを実行して、namespace 内のイベントを確認します。

```
$ oc get events -n <namespace> --sort-by=".metadata.creationTimestamp" 1
```

1 **--sort-by=".metadata.creationTimestamp"** フラグを追加すると、最新のイベントが出力の最後に配置されます。

2. オプション: 指定した namespace 内のイベントで十分な情報が得られない場合は、次のコマンドを実行して、クエリーの対象をすべての namespace に拡張します。

```
$ oc get events -A --sort-by=".metadata.creationTimestamp" 1
```

1 **--sort-by=".metadata.creationTimestamp"** フラグを指定すると、最新のイベントが出力の最後に配置されます。

クラスターからのすべてのイベントの結果をフィルタリングするには、**grep** コマンドを使用できます。たとえば、エラーを探している場合、エラーは出力の 2 つの異なるセクション (**TYPE** または **MESSAGE** セクション) に表示されることがあります。**grep** コマンドを使用すると、**error** や **failed** などのキーワードを検索できます。

3. たとえば、次のコマンドを実行して、**warning** または **error** を含むメッセージを検索します。

```
$ oc get events -A | grep -Ei "warning|error"
```

出力例

```
NAMESPACE   LAST SEEN   TYPE      REASON        OBJECT           MESSAGE
openshift   59s         Warning   FailedMount   pod/openshift-1  MountVolume.SetUp failed
for volume "v4-0-config-user-idp-0-file-data" : references non-existent secret key: test
```

4. オプション: イベントをクリーンアップして繰り返し発生しているイベントのみを表示するには、次のコマンドを実行して、関連する namespace 内のイベントを削除します。

```
$ oc delete events -n <namespace> --all
```

詳細は、「クラスターイベントの監視」を参照してください。

関連情報

- クラスターイベントの監視

16.2.2.5. Pod への接続

oc rsh コマンドを使用して、現在実行中の Pod に直接接続すると、その Pod のシェルを使用できます。

警告

低レイテンシーのアプリケーションを実行する Pod では、**oc rsh** コマンドを実行するとレイテンシーの問題が発生する可能性があります。**oc rsh** コマンドは、**oc debug** コマンドを使用してノードに接続できない場合にのみ使用してください。

手順

- 次のコマンドを実行して Pod に接続します。

```
$ oc rsh -n <namespace> busybox-1
```

詳細は、「oc rsh」および「実行中の Pod へのアクセス」を参照してください。

関連情報

- oc rsh
- 実行中の Pod へのアクセス

16.2.2.6. Pod のデバッグ

場合によっては、実稼働環境の Pod を直接操作するのを避けたほうがよいこともあります。

実行中のトラフィックを妨げないようにするには、元の Pod のコピーであるセカンダリー Pod を使用します。セカンダリー Pod は元の Pod と同じコンポーネントを使用しますが、セカンダリー Pod には実行中のトラフィックがありません。

手順

1. 次のコマンドを実行して、Pod を一覧表示します。

```
$ oc get pod
```

出力例

```
NAME       READY   STATUS    RESTARTS        AGE
busybox-1  1/1     Running   168 (34m ago)   7d
```

```
busybox-2   1/1     Running   119 (9m20s ago)   4d23h
busybox-3   1/1     Running   168 (43m ago)     7d
busybox-4   1/1     Running   168 (43m ago)     7d
```

2. 次のコマンドを実行して Pod をデバッグします。

```
$ oc debug -n <namespace> busybox-1
```

出力例

```
Starting pod/busybox-1-debug, command was: sleep 3600
Pod IP: 10.133.2.11
```

シェルプロンプトが表示されない場合は、Enter キーを押します。

詳細は、「oc debug」および「root アクセスでのデバッグ Pod の起動」を参照してください。

関連情報

- oc debug
- root アクセスでのデバッグ Pod の起動

16.2.2.7. Pod でのコマンドの実行

Pod に直接ログインせずに、Pod 上でコマンドまたはコマンドのセットを実行する場合は、**oc exec -it** コマンドを使用できます。Pod を素早く操作して、Pod からプロセス情報や出力情報を取得できます。一般的な使用例としては、スクリプト内で **oc exec -it** コマンドを実行して、レプリカセットまたはデプロイメント内の複数の Pod で同じコマンドを実行することが挙げられます。

警告

低レイテンシーのアプリケーションを実行する Pod では、**oc exec** コマンドによってレイテンシーの問題が発生する可能性があります。

手順

- Pod にログインせずに Pod でコマンドを実行するには、次のコマンドを実行します。

```
$ oc exec -it <pod> -- <command>
```

詳細は、「oc exec」および「コンテナー内でのリモートコマンドの実行」を参照してください。

関連情報

- oc exec
- コンテナーでのリモートコマンドの実行

16.2.3. クラスターのメンテナンス

通信事業者ネットワークでは、ベアメタルデプロイメントの性質上、特定の設定にさらに注意を払う必要があります。次のタスクを完了すると、より効果的にトラブルシューティングを行うことができます。

- 故障したハードウェアコンポーネントを監視する
- 定期的にクラスター Operator のステータスを確認する

注記

ハードウェアの監視は、ハードウェアベンダーに問い合わせて、特定のハードウェアに適したログツールを見つけてください。

16.2.3.1. クラスター Operator の確認

問題を早期に発見するために、クラスター Operator のステータスを定期的に確認します。

手順

- 次のコマンドを実行して、クラスター Operator のステータスを確認します。

```
$ oc get co
```

16.2.3.2. 失敗した Pod の監視

トラブルシューティングの時間を短縮するには、クラスター内の障害が発生した Pod を定期的に監視します。

手順

- 失敗した Pod を監視するには、次のコマンドを実行します。

```
$ oc get po -A | grep -Eiv 'complete|running'
```

16.2.4. セキュリティー

堅牢なクラスターセキュリティープロファイルを実装することは、回復力のある通信事業者ネットワークを構築する上で重要です。

16.2.4.1. Authentication

クラスター内にあるアイデンティティープロバイダーを特定します。サポートされているアイデンティティープロバイダーの詳細は、**認証および認可** の「サポートされるアイデンティティープロバイダー」を参照してください。

どのプロバイダーが設定されているのかがわかったら、**openshift-authentication** namespace を調べて、潜在的な問題があるかどうかを確認できます。

手順

1. 次のコマンドを実行して、**openshift-authentication** namespace 内のイベントを確認します。

```
$ oc get events -n openshift-authentication --sort-by='.metadata.creationTimestamp'
```

2. 次のコマンドを実行して、**openshift-authentication** namespace 内の Pod を確認します。

```
$ oc get pod -n openshift-authentication
```

3. オプション: 詳細情報が必要な場合は、次のコマンドを実行して、実行中のいずれかの Pod のログを確認します。

```
$ oc logs -n openshift-authentication <pod_name>
```

関連情報

- サポートされるアイデンティティープロバイダー

16.2.5. 証明書のメンテナンス

継続的なクラスター認証を実現するには、証明書のメンテナンスが欠かせません。一部の証明書はクラスター管理者が手動で更新する必要がありますが、その他の証明書はクラスターによって自動的に更新されます。

以下のリソースを使用して、OpenShift Container Platform の証明書とその管理方法を確認してください。

- Which OpenShift certificates do rotate automatically and which do not in Openshift 4.x?
- Checking etcd certificate expiry in OpenShift 4

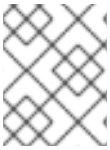
16.2.5.1. 管理者が手動で管理する証明書

次の証明書は、クラスター管理者が更新する必要があります。

- プロキシー証明書
- ユーザーがプロビジョニングする API サーバー証明書

16.2.5.1.1. プロキシー証明書の管理

プロキシー証明書を使用すると、ユーザーは、Egress 接続を行うときにプラットフォームコンポーネントによって使用される 1 つ以上のカスタム認証局 (CA) 証明書を指定できます。



注記

一部の CA には有効期限が設定されています。このような証明書は 2 年ごとに更新する必要がある場合があります。

要求される証明書を最初に設定しなかった場合は、いくつかの方法で証明書の有効期限を決定できます。ほとんどのクラウドネイティブネットワーク機能 (CNF) が使用する証明書は、ブラウザーベースの接続用に特別に設計されたものではありません。したがって、デプロイメントの **ConfigMap** オブジェクトから証明書を取得する必要があります。

手順

- 有効期限を取得するには、証明書ファイルに対して次のコマンドを実行します。

```
$ openssl x509 -enddate -noout -in <cert_file_name>.pem
```

プロキシー証明書を更新する方法と時期を確認する方法の詳細は、**セキュリティーおよびコンプライアンス** の「プロキシー証明書」を参照してください。

関連情報

- プロキシー証明書

16.2.5.1.2. ユーザーがプロビジョニングする API サーバー証明書

API サーバーには、クラスター外部のクライアントから **api.<cluster_name>.<base_domain>** でアクセスできます。クライアントに別のホスト名で API サーバーにアクセスさせたり、クラスター管理の認証局 (CA) 証明書をクライアントに配布せずに API サーバーにアクセスさせたりする必要が生じる場合があります。コンテンツを提供するときに API サーバーが使用するカスタムのデフォルト証明書を設定する必要があります。

詳細は、**セキュリティーおよびコンプライアンス** の「API サーバー用のユーザーがプロビジョニングする証明書」を参照してください。

関連情報

- ユーザーがプロビジョニングする API サーバー証明書

16.2.5.2. クラスターによって管理される証明書

クラスターによって管理される証明書は、ログで問題が検出された場合にのみ確認する必要があります。次の証明書はクラスターによって自動的に管理されます。

- サービス CA 証明書
- ノード証明書
- ブートストラップ証明書
- etcd 証明書
- OLM 証明書
- Machine Config Operator 証明書
- モニタリングおよびクラスターロギング Operator コンポーネント証明書
- コントロールプレーンの証明書
- Ingress 証明書

関連情報

- サービス CA 証明書
- ノード証明書

- ブートストラップ証明書
- etcd 証明書
- OLM 証明書
- Machine Config Operator 証明書
- モニタリングおよびクラスターロギング Operator コンポーネント証明書
- コントロールプレーンの証明書
- Ingress 証明書

16.2.5.2.1. etcd によって管理される証明書

etcd 証明書は、etcd メンバーのピア間の暗号化された通信と、暗号化されたクライアントトラフィックに使用されます。すべてのノードとすべてのサービス間の通信が最新であれば、証明書はクラスター内で自動的に更新されます。したがって、etcd 証明書の有効期限に近い特定期間中にクラスターがコンポーネント間の通信を失う可能性がある場合は、事前に証明書を更新することを推奨します。たとえば、ノードが異なるタイミングで再起動するため、アップグレード中に通信が失われる可能性があります。

- 次のコマンドを実行して、etcd 証明書を手動で更新できます。

```
$ for each in $(oc get secret -n openshift-etcd | grep "kubernetes.io/tls" | grep -e \
"etcd-peer\|etcd-serving" | awk '{print $1}'); do oc get secret $each -n openshift-etcd -o \
jsonpath="{.data.tls\.crt}" | base64 -d | openssl x509 -noout -enddate; done
```

etcd 証明書の更新の詳細は、Checking etcd certificate expiry in OpenShift 4 を参照してください。
etcd 証明書の詳細は、**セキュリティーおよびコンプライアンス** の「etcd 証明書」を参照してください。

関連情報

- etcd 証明書

16.2.5.2.2. ノード証明書

ノード証明書は自己署名証明書です。つまり、クラスターによって署名され、ブートストラッププロセスによって生成された内部認証局 (CA) から発行されたものです。

クラスターがインストールされると、クラスターはノード証明書を自動的に更新します。

詳細は、**セキュリティーおよびコンプライアンス** の「ノード証明書」を参照してください。

関連情報

- ノード証明書

16.2.5.2.3. サービス CA 証明書

service-ca は、OpenShift Container Platform クラスターがデプロイされるときに自己署名認証局 (CA) を作成する Operator です。これにより、ユーザーは手動で証明書を作成せずに、デプロイメントに証明書を追加できます。サービス CA 証明書は自己署名証明書です。

詳細は、**セキュリティーおよびコンプライアンス** の「サービス CA 証明書」を参照してください。

関連情報

- サービス CA 証明書

16.2.6. Machine Config Operator

Machine Config Operator は、クラスター管理者に有用な情報を提供し、ベアメタルホスト上で直接実行されているコンポーネントを制御します。

Machine Config Operator は、クラスター内の各ノードグループを区別し、コントロールプレーンノードとワーカーノードを異なる設定で実行できるようにします。これらのノードグループは、**MachineConfigPool** (**mcp**) グループと呼ばれるワーカー Pod またはアプリケーション Pod を実行します。同じマシン設定が、クラスター内のすべてのノードまたは 1 つの MCP にのみ適用されます。

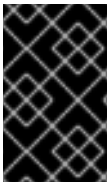
通信事業者コアクラスターで MCP を適用する方法と理由の詳細は、更新前にノードに MachineConfigPool ラベルを適用する を参照してください。

Machine Config Operator の詳細は、Machine Config Operator を参照してください。

16.2.6.1. Machine Config Operator の目的

Machine Config Operator (MCO) は、カーネルと kubelet 間にあるすべてのものを含む、Red Hat Enterprise Linux CoreOS (RHCOS) とコンテナーランタイムの設定と更新を管理および適用します。ほとんどの通信事業者はベアメタルハードウェア上で運用し、何らかのハードウェアアクセラレーターまたはカーネルの変更を使用しているため、RHCOS の管理は重要です。MCO は各ノードとそれに適用される内容を監視するため、マシン設定を RHCOS に手動で適用すると問題が発生する可能性があります。

このような細かいコンポーネントと、MCO がクラスターを効果的に管理するのにどのように役立つかを考慮する必要があります。



重要

ワーカーノードまたはコントロールプレーンノードに対する変更は、すべて MCO を使用して実行する必要があります。RHCOS またはノードファイルを手動で変更しないでください。

16.2.6.2. 複数のマシン設定ファイルを同時に適用する

クラスター内のノードグループ (マシン設定プール (MCP) とも呼ばれます) のマシン設定を変更する必要がある場合、複数の異なるマシン設定ファイルを使用して変更を適用する必要があることがあります。マシン設定ファイルを適用するには、ノードを再起動する必要があります。各マシン設定ファイルがクラスターに適用されると、そのマシン設定ファイルの影響を受けるすべてのノードが再起動します。

マシン設定ファイルごとにノードが再起動するのを防ぐには、新しいマシン設定ファイルによって更新される各 MCP を一時停止して、すべての変更を同時に適用します。

手順

1. 次のコマンドを実行して、影響を受ける MCP を一時停止します。

```
$ oc patch mcp/<mcp_name> --type merge --patch '{"spec":{"paused":true}}'
```

2. すべてのマシン設定の変更をクラスターに適用したら、次のコマンドを実行します。

```
$ oc patch mcp/<mcp_name> --type merge --patch '{"spec":{"paused":false}}'
```

これにより、MCP 内のノードが新しい設定で再起動できるようになります。

16.2.7. ベアメタルノードのメンテナンス

ノードに接続すると、一般的なトラブルシューティングを実行できます。ただし、場合によっては、特定のハードウェアコンポーネントに対してトラブルシューティングやメンテナンスのタスクを実行する必要があります。このセクションでは、ハードウェアのメンテナンスを実行するために必要なトピックを説明します。

16.2.7.1. クラスター内のベアメタルノードに接続する

ベアメタルクラスターノードに接続すると、一般的なメンテナンスタスクを実行できます。

注記

ホストオペレーティングシステムからクラスターノードを設定することは推奨されておらず、サポートもされていません。

ノードのトラブルシューティングを行う際には、次のタスクを実行できます。

- ノードからログを取得する
- デバッグを使用する
- SSH を使用してノードに接続する

重要

SSH は、**oc debug** コマンドを使用してノードに接続できない場合にのみ使用してください。

手順

1. 次のコマンドを実行して、ノードからログを取得します。

```
$ oc adm node-logs <node_name> -u crio
```

2. 次のコマンドを実行してデバッグを使用します。

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

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。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
```

出力

```
You are now logged in as root on the node
```

4. オプション: 次のコマンドを実行して、SSH を使用してノードに接続します。

```
$ ssh core@<node_name>
```

16.2.7.2. クラスター内の Pod にアプリケーションを移動する

計画的なハードウェアメンテナンスの場合、Pod のワークロードに影響を与えずに、アプリケーション Pod をクラスター内の他のノードに移動する方法を検討する必要があります。

手順

- 次のコマンドを実行して、ノードをスケジュール対象外としてマークします。

```
$ oc adm cordon <node_name>
```

ノードがスケジュール対象外の場合、そのノードで Pod をスケジュールすることはできません。詳細は、「ノードの操作」を参照してください。

注記

CNF アプリケーションを移動する場合、アンチアフィニティーと Pod Disruption Budget 用に、クラスター内に十分な追加ワーカーノードがあるかどうかを事前に確認する必要がある場合があります。

関連情報

- ノードの使用

16.2.7.3. DIMM メモリーの交換

デュアルインラインメモリーモジュール (DIMM) の問題が発生することがあるのは、サーバーの再起動後だけです。このような問題はログファイルで確認できます。

標準の再起動を実行してもサーバーが起動しない場合は、DIMM メモリーに障害があることを示すメッセージがコンソールに表示されます。その場合、障害のある DIMM を確認し、残りのメモリーが十分であれば再起動を続行できます。その後、障害のある DIMM を交換するためのメンテナンスウィンドウをスケジュールすることができます。

場合によっては、イベントログのメッセージにメモリーモジュールの不良が示されることがあります。このような場合は、サーバーを再起動する前にメモリーの交換をスケジュールすることができます。

関連情報

- OpenShift Container Platform ストレージの概要

16.2.7.4. ディスクの交換

ハードウェアまたはソフトウェアの Redundant Array of Independent Disks (RAID) を通じてノードにディスク冗長性が設定されていない場合は、次の点を確認する必要があります。

- ディスクに実行中の Pod イメージが含まれているかどうか
- ディスクに Pod の永続データが含まれているかどうか

詳細は、**ストレージ** の「OpenShift Container Platform ストレージの概要」を参照してください。

16.2.7.5. クラスターネットワークカードの交換

ネットワークカードを交換すると、MAC アドレスが変更されます。MAC アドレスは、DHCP または SR-IOV Operator 設定、ルーター設定、ファイアウォールルール、またはアプリケーションのクラウドネイティブネットワーク機能 (CNF) 設定の一部である場合があります。ネットワークカードを交換した後、ノードをオンラインに戻す前に、これらの設定が最新であることを確認する必要があります。

重要

ネットワーク内で MAC アドレスを変更するための具体的な手順が整備されていない場合は、ネットワーク管理者またはネットワークハードウェアベンダーに問い合わせてください。

16.3. 可観測性

16.3.1. OpenShift Container Platform における可観測性

OpenShift Container Platform では、プラットフォームとプラットフォーム上で実行されているワークロードの両方から、パフォーマンスメトリクスやログなどの大量のデータが生成されます。管理者は、さまざまなツールを使用して、利用可能なすべてのデータを収集および分析できます。以下は、可観測性スタックを設定するシステムエンジニア、アーキテクト、および管理者向けのベストプラクティスの概要です。

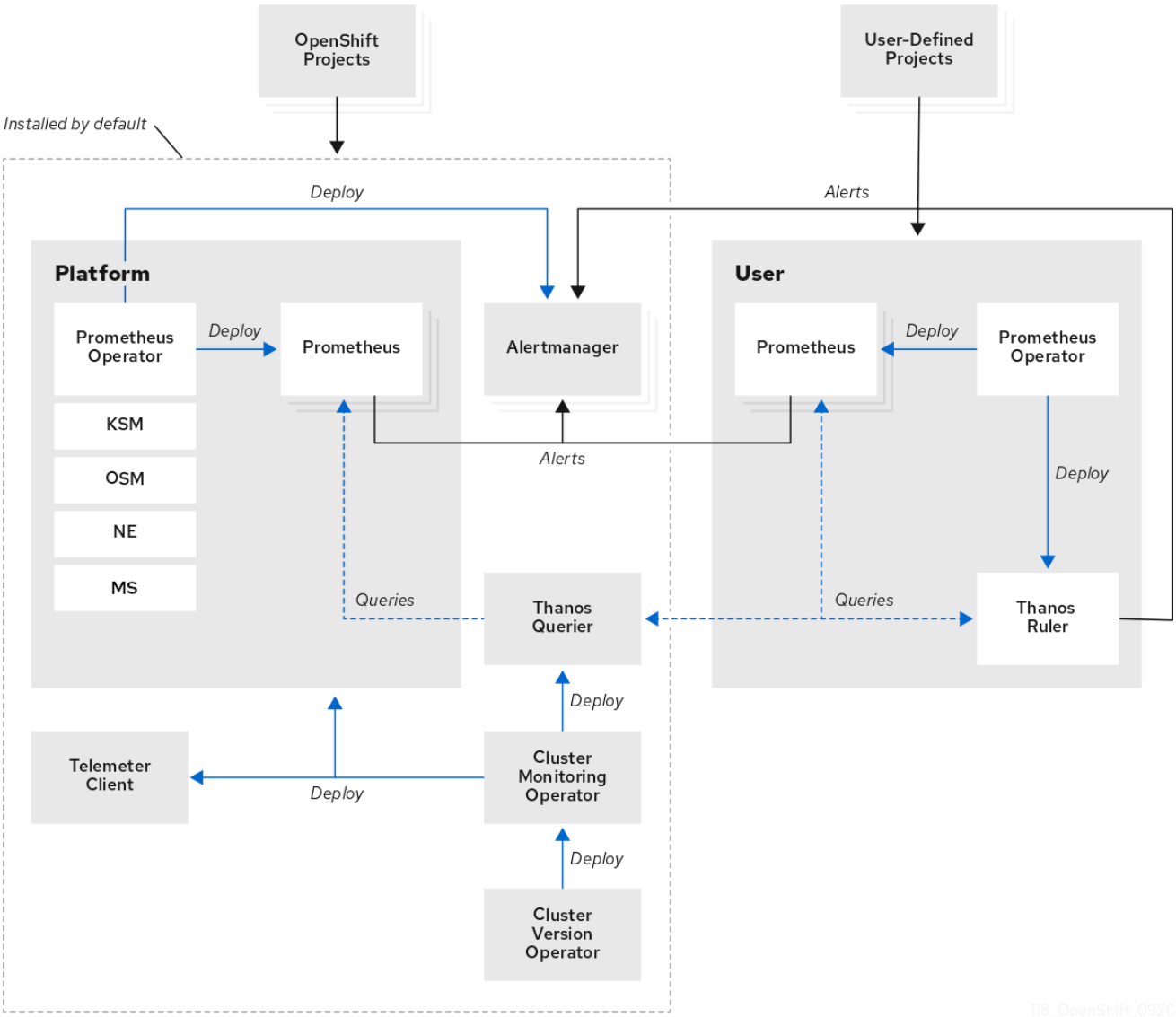
特に明記のない限り、このドキュメントの内容はエッジデプロイメントとコアデプロイメントの両方を表しています。

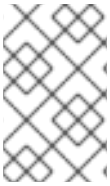
16.3.1.1. モニタリングスタックについて

モニタリングスタックは次のコンポーネントを使用します。

- Prometheus は、OpenShift Container Platform コンポーネントおよびワークロードからメトリクスを収集して分析します (そのように設定されている場合)。
- Alertmanager は、アラートのルーティング、グループ化、およびサイレンスを処理する Prometheus のコンポーネントです。
- Thanos はメトリクスの長期保存を処理します。

図16.2 OpenShift Container Platform モニタリングアーキテクチャー





注記

シングルノード OpenShift クラスターの場合、クラスターは分析と保持のためにすべてのメトリクスをハブクラスターに送信するため、Alertmanager と Thanos を無効にする必要があります。

関連情報

- OpenShift Container Platform モニタリングについて
- コアプラットフォームモニタリングの最初のステップ

16.3.1.2. 主要なパフォーマンスメトリクス

システムによっては、利用可能な測定値が数百種類ある場合があります。

注目すべき重要なメトリクスは次のとおりです。

- **etcd** 応答時間

- API 応答時間
- Pod の再起動とスケジュール
- リソースの使用状況
- OVN の健全性
- クラスター Operator の全体的な健全性

原則として、あるメトリクスが重要であると判断した場合は、それに対するアラートを設定することを推奨します。

注記

次のコマンドを実行すると、利用可能なメトリクスを確認できます。

```
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -qsk
http://localhost:9090/api/v1/metadata | jq '.data
```

16.3.1.2.1. PromQL のクエリー例

次の表は、OpenShift Container Platform コンソールを使用してメトリクスクエリーブラウザーで調べることができるクエリーの一部を示しています。

注記

コンソールの URL は https://<OpenShift Console FQDN>/monitoring/query-browser です。以下のコマンドを実行して、OpenShift Console FQDN を取得できます。

```
$ oc get routes -n openshift-console console -o jsonpath='{.status.ingress[0].host}'
```

表16.1 ノードのメモリーと CPU の使用率

メトリクス	クエリー
ノード別の CPU 要求 (%)	**sum by (node) (sum_over_time(kube_pod_container_resource_requests{resource="cpu"}[60m]))/sum by (node) (sum_over_time(kube_node_status_allocatable{resource="cpu"}[60m])) *100**
クラスター全体の CPU 使用率 (%)	**sum by (managed_cluster) (sum_over_time(kube_pod_container_resource_requests{resource="memory"}[60m]))/sum by (managed_cluster) (sum_over_time(kube_node_status_allocatable{resource="cpu"}[60m])) *100**

メトリクス	クエリー
ノード別のメモリー要求 (%)	**sum by (node) (sum_over_time(kube_pod_container_resource_requests{resource="memory"} [60m]))/sum by (node) (sum_over_time(kube_node_status_allocatable{resource="memory"}[60m])) *100**
クラスター全体のメモリー使用率 (%)	**(1-(sum by (managed_cluster) (avg_over_timenode_memory_MemAvailable_bytes[60m]))/sum by (managed_cluster) (avg_over_time(kube_node_status_allocatable{resource="memory"}[60m])))*100**

表16.2 動詞別の API レイテンシー

メトリクス	クエリー
GET	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver=~"kube-apiserver|openshift-apiserver", verb="GET"} [60m])))**
PATCH	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver="kube-apiserver|openshift-apiserver", verb="PATCH"}[60m])))**
POST	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver="kube-apiserver|openshift-apiserver", verb="POST"}[60m])))**
LIST	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver="kube-apiserver|openshift-apiserver", verb="LIST"} [60m])))**

メトリクス	クエリー
PUT	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver="kube-apiserver|openshift-apiserver", verb="PUT"}[60m])))**
DELETE	**histogram_quantile (0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver="kube-apiserver|openshift-apiserver", verb="DELETE"}[60m])))**
合計	**histogram_quantile(0.99, sum by (le,managed_cluster) (sum_over_time(apiserver_request_duration_seconds_bucket{apiserver=~"(openshift-apiserver|kube-apiserver)", verb!="WATCH"}[60m])))**

表16.3 etcd

メトリクス	クエリー
fsync 99 パーセンタイルレイテンシー (インスタンスあたり)	**histogram_quantile(0.99, rate(etcd_disk_wal_fsync_duration_seconds_bucket[2m]))**
fsync 99 パーセンタイルレイテンシー (クラスターあたり)	**sum by (managed_cluster) (histogram_quantile(0.99, rate(etcd_disk_wal_fsync_duration_seconds_bucket[60m])))**
リーダー選出	**sum(rate(etcd_server_leader_changes_seen_total[1440m]))**
ネットワークレイテンシー	**histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[5m]))**

表16.4 Operator の健全性

メトリクス	クエリー
デグレード状態の Operator	**sum by (managed_cluster, name) (avg_over_time(cluster_operator_conditions{ condition="Degraded", name!="version"} [60m]))**
クラスターあたりのデグレード状態の Operator 総数	**sum by (managed_cluster) (avg_over_time(cluster_operator_conditions{ condition="Degraded", name!="version"} [60m]))**

16.3.1.2.2. メトリクスの保存に関する推奨事項

デフォルトでは、Prometheus は保存されたメトリクスを永続ストレージにバックアップしません。Prometheus Pod を再起動すると、すべてのメトリクスデータが失われます。プラットフォームで使用可能なバックエンドストレージを使用するようにモニタリングスタックを設定する必要があります。Prometheus の高い IO 要求を満たすには、ローカルストレージを使用する必要があります。

通信事業者コアクラスターの場合、Prometheus の永続ストレージに Local Storage Operator を使用できます。

ブロック、ファイル、およびオブジェクトストレージ用の Ceph クラスターをデプロイする Red Hat OpenShift Data Foundation (ODF) も、通信事業者コアクラスターの候補として適しています。

RAN シングルノード OpenShift またはファーエッジクラスターのシステムリソース要件を低く抑えるには、モニタリングスタック用のバックエンドストレージをプロビジョニングしないでください。このようなクラスターは、すべてのメトリクスをハブクラスターに転送します。そこにサードパーティーのモニタリングプラットフォームをプロビジョニングできます。

関連情報

- 管理者としてのメトリクスへのアクセス
- ローカルボリュームを使用した永続ストレージ

16.3.1.3. エッジのモニタリング

シングルノード OpenShift をエッジに配置することで、プラットフォームコンポーネントのフットプリントが最小限に抑えられます。次の手順は、モニタリングフットプリントが小さいシングルノード OpenShift ノードを設定する方法の例です。

前提条件

- Red Hat Advanced Cluster Management (RHACM) を使用する環境で、可観測性サービスが有効になっている。
- ハブクラスターで Red Hat OpenShift Data Foundation (ODF) が実行されている。

手順

1. 次の例のように、**ConfigMap** CR を作成し、**monitoringConfigMap.yaml** として保存します。
   ■

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

2. シングルノード OpenShift で、次のコマンドを実行して **ConfigMap** CR を適用します。

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

3. 次の例のように、**NameSpace** CR を作成し、**monitoringNamespace.yaml** として保存します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: open-cluster-management-observability
```

4. ハブクラスターで、次のコマンドを実行して、ハブクラスターに **Namespace** CR を適用します。

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

5. 次の例のように、**ObjectBucketClaim** CR を作成し、**monitoringObjectBucketClaim.yaml** として保存します。

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: multi-cloud-observability
  namespace: open-cluster-management-observability
spec:
  storageClassName: openshift-storage.noobaa.io
  generateBucketName: acm-multi
```

6. ハブクラスターで、次のコマンドを実行して **ObjectBucketClaim** CR を適用します。

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

7. 次の例のように、**Secret** CR を作成し、**monitoringSecret.yaml** として保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: multiclusterhub-operator-pull-secret
```

```
  namespace: open-cluster-management-observability
stringData:
  .dockerconfigjson: 'PULL_SECRET'
```

8. ハブクラスターで、次のコマンドを実行して **Secret** CR を適用します。

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

9. 次のコマンドを実行して、ハブクラスターから NooBaa サービスのキーとバックエンドバケット名を取得します。

```
$ NOOBAA_ACCESS_KEY=$(oc get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_ACCESS_KEY_ID|@base64d')
```

```
$ NOOBAA_SECRET_KEY=$(oc get secret noobaa-admin -n openshift-storage -o json | jq -r
'.data.AWS_SECRET_ACCESS_KEY|@base64d')
```

```
$ OBJECT_BUCKET=$(oc get objectbucketclaim -n open-cluster-management-observability
multi-cloud-observability -o json | jq -r .spec.bucketName)
```

10. 次の例のように、バケットストレージ用の **Secret** CR を作成し、**monitoringBucketSecret.yaml** として保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-object-storage
  namespace: open-cluster-management-observability
type: Opaque
stringData:
  thanos.yaml: |
    type: s3
    config:
      bucket: ${OBJECT_BUCKET}
      endpoint: s3.openshift-storage.svc
      insecure: true
      access_key: ${NOOBAA_ACCESS_KEY}
      secret_key: ${NOOBAA_SECRET_KEY}
```

11. ハブクラスターで、次のコマンドを実行して **Secret** CR を適用します。

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

12. 次の例のように、**MultiClusterObservability** CR を作成し、**monitoringMultiClusterObservability.yaml** として保存します。

```
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
spec:
  advanced:
    retentionConfig:
```

```
    blockDuration: 2h
    deleteDelay: 48h
    retentionInLocal: 24h
    retentionResolutionRaw: 3d
  enableDownsampling: false
  observabilityAddonSpec:
    enableMetrics: true
    interval: 300
  storageConfig:
    alertmanagerStorageSize: 10Gi
    compactStorageSize: 100Gi
    metricObjectStorage:
      key: thanos.yaml
      name: thanos-object-storage
    receiveStorageSize: 25Gi
    ruleStorageSize: 10Gi
    storeStorageSize: 25Gi
```

13. ハブクラスターで、次のコマンドを実行して **MultiClusterObservability** CR を適用します。

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

検証

1. 次のコマンドを実行して、namespace 内のルートと Pod を確認し、サービスがハブクラスターにデプロイされていることを確認します。

```
$ oc get routes,pods -n open-cluster-management-observability
```

出力例

```
NAME                                     HOST/PORT
PATH      SERVICES                       PORT         TERMINATION          WILDCARD
route.route.openshift.io/alertmanager    alertmanager-open-cluster-management-
observability.cloud.example.com          /api/v2   alertmanager                 oauth-proxy
reencrypt/Redirect   None
route.route.openshift.io/grafana         grafana-open-cluster-management-
observability.cloud.example.com                    grafana                      oauth-proxy
reencrypt/Redirect   None (1)
route.route.openshift.io/observatorium-api   observatorium-api-open-cluster-management-
observability.cloud.example.com            observability-observatorium-api   public
passthrough/None     None
route.route.openshift.io/rbac-query-proxy    rbac-query-proxy-open-cluster-management-
observability.cloud.example.com            rbac-query-proxy                 https
reencrypt/Redirect   None

NAME                                          READY   STATUS    RESTARTS   AGE
pod/observability-alertmanager-0              3/3     Running   0          1d
pod/observability-alertmanager-1              3/3     Running   0          1d
pod/observability-alertmanager-2              3/3     Running   0          1d
pod/observability-grafana-685b47bb47-dq4cw                3/3     Running   0          1d
<...snip…>
```

```
pod/observability-thanos-store-shard-0-0                1/1     Running   0          1d
pod/observability-thanos-store-shard-1-0                1/1     Running   0          1d
pod/observability-thanos-store-shard-2-0                1/1     Running   0          1d
```

1 リスト表示される Grafana ルートからダッシュボードにアクセスできます。これを使用して、すべてのマネージドクラスターのメトリクスを表示できます。

{rh-rhacm-title} の可観測性の詳細は、可 観測性 を参照してください。

16.3.1.4. アラート

OpenShift Container Platform には多数のアラートルールが含まれています。ルールはリリースごとに変更される可能性があります。

16.3.1.4.1. デフォルトのアラートの表示

クラスター内のすべてのアラートルールを確認するには、次の手順を使用します。

手順

- クラスター内のすべてのアラートルールを確認するには、次のコマンドを実行します。

```
$ oc get cm -n openshift-monitoring prometheus-k8s-rulefiles-0 -o yaml
```

ルールには説明を含めることができ、追加情報や軽減策へのリンクを提供できます。たとえば、**etcdHighFsyncDurations** のルールは次のとおりです。

```
    - alert: etcdHighFsyncDurations
      annotations:
        description: 'etcd cluster "{{ $labels.job }}": 99th percentile fsync durations
          are {{ $value }}s on etcd instance {{ $labels.instance }}.'
        runbook_url: https://github.com/openshift/runbooks/blob/master/alerts/cluster-etcd-
operator/etcdHighFsyncDurations.md
        summary: etcd cluster 99th percentile fsync durations are too high.
      expr: |
        histogram_quantile(0.99,
rate(etcd_disk_wal_fsync_duration_seconds_bucket{job=~".*etcd.*"}[5m]))
        > 1
      for: 10m
      labels:
        severity: critical
```

16.3.1.4.2. アラート通知

アラートは OpenShift Container Platform コンソールで表示できますが、アラートを転送する外部レシーバーを設定することが管理者には推奨されます。OpenShift Container Platform は、次のレシーバータイプをサポートしています。

- PagerDuty: サードパーティーのインシデント対応プラットフォーム
- Webhook: POST リクエストを介してアラートを受信し、必要なアクションを実行できる任意の API エンドポイント

- Email: 指定されたアドレスにメールを送信する
- Slack: Slack チャネルまたは個々のユーザーに通知を送信する

関連情報

- アラートの管理

16.3.1.5. ワークロードのモニタリング

デフォルトでは、OpenShift Container Platform はアプリケーションワークロードのメトリクスを収集しません。ワークロードメトリクスを収集するようにクラスターを設定できます。

前提条件

- クラスターのワークロードメトリクスを収集するためのエンドポイントを定義した。

手順

1. 次の例のように、**ConfigMap** CR を作成し、**monitoringConfigMap.yaml** として保存します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true (1)
```

(1) **true** に設定してワークロードのモニタリングを有効にします。

2. 次のコマンドを実行して **ConfigMap** CR を適用します。

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

3. 次の例のように、**ServiceMonitor** CR を作成し、**monitoringServiceMonitor.yaml** として保存します。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: ui
  name: myapp
  namespace: myns
spec:
  endpoints: (1)
  - interval: 30s
    port: ui-http
    scheme: http
    path: /healthz (2)
```

```
  selector:
    matchLabels:
      app: ui
```

1. エンドポイントを使用してワークロードメトリクスを定義します。
2. Prometheus はデフォルトでパス **/metrics** をスクレイピングします。ここでカスタムパスを定義できます。

4. 次のコマンドを実行して、**ServiceMonitor** CR を適用します。

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

Prometheus はデフォルトでパス **/metrics** をスクレイピングしますが、カスタムパスを定義することもできます。重要であると判断したメトリクスとともに、このエンドポイントをスクレイピング用に公開するかどうかは、アプリケーションのベンダー次第です。

16.3.1.5.1. ワークロードのアラートの作成

クラスター上のユーザーワークロードに対するアラートを有効にできます。

手順

1. 次の例のように、**ConfigMap** CR を作成し、**monitoringConfigMap.yaml** として保存します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true 1
# ...
```

1. **true** に設定してワークロードのモニタリングを有効にします。

2. 次のコマンドを実行して **ConfigMap** CR を適用します。

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

3. 次の例のように、アラートルール用の YAML ファイル (**monitoringAlertRule.yaml**) を作成します。

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: myapp-alert
  namespace: myns
spec:
  groups:
  - name: example
```

```
    rules:
    - alert: InternalErrorsAlert
      expr: flask_http_request_total{status="500"} > 0
# ...
```

4. 次のコマンドを実行してアラートルールを適用します。

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

関連情報

- ServiceMonitor[monitoring.coreos.com/v1]
- ユーザー定義プロジェクトのモニタリングの有効化
- ユーザー定義プロジェクトのアラートルールの管理

16.4. セキュリティー

16.4.1. セキュリティーの基礎

セキュリティーは、特にクラウドネイティブネットワーク機能 (CNF) を実行する場合、OpenShift Container Platform 上の通信事業者デプロイメントの重要な要素となります。

セキュリティー上の重要な考慮事項を遵守することで、通信事業者 (telco) 環境における高帯域幅ネットワークデプロイメントのセキュリティーを強化できます。このような標準やベストプラクティスを実装すると、通信事業者固有のユースケースにおけるセキュリティーを強化できます。

16.4.1.1. RBAC の概要

ロールベースアクセス制御 (RBAC) オブジェクトは、ユーザーがプロジェクト内で所定のアクションを実行することが許可されるかどうかを決定します。

これにより、プラットフォーム管理者はクラスターロールおよびバインディングを使用して、OpenShift Container Platform プラットフォーム自体およびすべてのプロジェクトへの各種のアクセスレベルを持つユーザーを制御できます。

開発者はローカルロールとバインディングを使用して、プロジェクトにアクセスできるユーザーを制御できます。認可は認証とは異なる手順であることに注意してください。認証はアクションを実行するユーザーのアイデンティティーの判別により密接に関連しています。

認可は、次の認可オブジェクトを使用して管理されます。

ルール

特定のオブジェクトに対して許可されるアクションのセットです。たとえば、ルールによって、ユーザーまたはサービスアカウントが Pod を作成できるかどうかを決定できます。各ルールで、API リソース、その API 内のリソース、および許可されるアクションを指定します。

ロール

ユーザーまたはグループが実行できるアクションを定義するルールの集まりです。ルールを複数のユーザーまたはグループに関連付けたり、バインドしたりできます。ロールファイルに、そのロールに許可されるアクションおよびリソースを指定するルールを 1 つ以上含めることができます。ロールは次のタイプに分類されます。

- クラスターロール: クラスターレベルでクラスターロールを定義できます。クラスターロールは 1 つの namespace に結び付けられません。ユーザー、グループ、またはサービスアカウントにクラスターロールをバインドすると、すべての namespace または特定の namespace に適用できます。

- プロジェクトロール: 特定の namespace 内にプロジェクトロールを作成できます。プロジェクトロールはその namespace にのみ適用されます。特定のユーザーに権限を割り当てて、ロールとロールバインディングをそのロールの namespace 内に作成することで、他の namespace に影響を与えないようにすることができます。

バインディング

ロールを使用したユーザーやグループ間の関連付けです。ロールバインディングを作成すると、ロール内のルールを特定のユーザー ID またはグループと結び付けることができます。これにより、ロールとユーザーまたはグループが組み合わせられ、ユーザーまたはグループが実行できるアクションが定義されます。

注記

ユーザーまたはグループには複数のロールをバインドできます。

RBAC の詳細は、「RBAC を使用した権限の定義および適用」を参照してください。

RBAC の運用上の考慮事項
運用上のオーバーヘッドを削減するには、複数のクラスターにわたって個々のユーザー ID を処理するのではなく、グループを通じてアクセスを管理することが重要です。組織レベルでグループを管理することで、アクセス制御を合理化し、組織全体の管理を簡素化できます。

関連情報

- RBAC を使用した権限の定義および適用

16.4.1.2. セキュリティーアカウントの概要

サービスアカウントは、コンポーネントが API に直接アクセスできるようにする OpenShift Container Platform アカウントです。サービスアカウントは各プロジェクトに存在する API オブジェクトです。サービスアカウントは、一般ユーザーの認証情報を共有せずに API アクセスをより柔軟に制御する方法を提供します。

サービスアカウントを使用すると、Pod にロールベースのアクセス制御 (RBAC) を適用できます。Pod やデプロイメントなどのワークロードにサービスアカウントを割り当てることで、さまざまなレジストリーからのプルなど、追加の権限を付与できます。これにより、より低い権限をサービスアカウントに割り当てて、そのアカウント配下で実行される Pod のセキュリティー関連のフットプリントを削減することもできます。

サービスアカウントの詳細は、「サービスアカウントの概要および作成」を参照してください。

関連情報

- サービスアカウントの概要および作成

16.4.1.3. アイデンティティープロバイダーの設定

アイデンティティープロバイダーの設定は、クラスターでユーザーを設定する際の最初のステップです。アイデンティティープロバイダーを使用すると、組織レベルでグループを管理できます。

アイデンティティープロバイダーは、クラスターレベルではなく組織レベルで管理される特定のユーザーグループを取得できます。これにより、組織の確立されたプラクティスに準拠したグループにユーザーを追加したり、グループからユーザーを削除したりできます。

注記

変更をクラスターに取り込むには、頻繁に実行される cron ジョブを設定する必要があります。

アイデンティティープロバイダーを使用すると、組織内の特定グループのアクセスレベルを管理できます。たとえば、アクセスレベルを管理するために、次の操作を実行できます。

- クラスターレベルの権限を必要とするチームに **cluster-admin** ロールを割り当てる。
- アプリケーション管理者に、それぞれのプロジェクトのみを管理するための特定の権限を付与する。
- 運用チームに、クラスター全体の **view** アクセス権を提供し、変更を許可することなく、監視できるようにする。

アイデンティティープロバイダーの設定の詳細は、「アイデンティティープロバイダー設定について」を参照してください。

関連情報

- アイデンティティープロバイダー設定について

16.4.1.4. kubeadmin ユーザーを cluster-admin ユーザーに置き換える

デフォルトでは、すべてのクラスターに **cluster-admin** 権限を持つ **kubeadmin** ユーザーが作成されます。クラスターのセキュリティーを強化するには、`kubeadmin` ユーザーを **cluster-admin** ユーザーに置き換えてから、**kubeadmin** ユーザーを無効にするか削除します。

前提条件

- **cluster-admin** 権限を持つユーザーを作成した。
- OpenShift CLI (**oc**) がインストールされている。
- セキュアに保存するための仮想 vault への管理アクセス権がある。

手順

1. **htpasswd** アイデンティティープロバイダーを使用して、緊急用の **cluster-admin** ユーザーを作成します。詳細は、「htpasswd 認証について」を参照してください。
2. 次のコマンドを実行して、新しいユーザーに **cluster-admin** 権限を割り当てます。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <emergency_user>
```

3. 緊急用ユーザーのアクセス権を検証します。

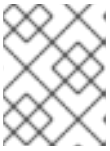
a. 新しい緊急用ユーザーを使用してクラスターにログインします。

b. 次のコマンドを実行して、ユーザーに **cluster-admin** 権限があることを確認します。

```
$ oc whoami
```

出力に緊急用ユーザーの ID が表示されることを確認します。

4. 緊急用ユーザーのパスワードまたは認証キーを仮想 vault にセキュアに保存します。



注記

機密性の高い認証情報の保護に関する組織のベストプラクティスに従ってください。

5. 次のコマンドを実行して、**kubeadmin** ユーザーを無効にするか削除し、セキュリティーリスクを軽減します。

```
$ oc delete secrets kubeadmin -n kube-system
```

関連情報

- htpasswd 認証について

16.4.1.5. 通信事業者 CNF のセキュリティーに関する考慮事項

通信事業者のワークロードは膨大な量の機密データを処理するため、高い信頼性が求められます。1 つのセキュリティー脆弱性が、クラスター全体に及ぶ広範な侵害につながる可能性があります。シングルノード OpenShift クラスター上で多数のコンポーネントが実行されている場合、侵害が拡大するのを防ぐために各コンポーネントを保護する必要があります。通信事業者のネットワークの信頼性を維持し、脆弱性を回避するには、すべてのコンポーネントを含むインフラストラクチャー全体のセキュリティーを確保することが不可欠です。

通信事業者にとっては、以下の主要なセキュリティー機能が欠かせません。

- Security Context Constraints (SCC): OpenShift クラスター内の Pod セキュリティーをきめ細かく制御します。
- Pod Security Admission (PSA): Kubernetes ネイティブの Pod セキュリティーコントロール。
- 暗号化: 高スループットのネットワーク環境でデータの機密性を確保します。

16.4.1.6. Kubernetes と OpenShift Container Platform における Pod セキュリティーの発展

Kubernetes では当初、Pod のセキュリティーが制限されていました。OpenShift Container Platform を Kubernetes と統合したときに、Red Hat は Security Context Constraints (SCC) によって Pod のセキュリティーを強化しました。同様の機能として、Kubernetes バージョン 1.3 で **PodSecurityPolicy** (PSP) が導入されました。しかし、Kubernetes バージョン 1.21 で Pod Security Admission (PSA) が導入されたことにより、Kubernetes バージョン 1.25 で PSP は非推奨になりました。

PSA は、OpenShift Container Platform でも、バージョン 4.11 で利用可能になりました。PSA は、Pod のセキュリティーを強化しますが、通信事業者のユースケースに現在も必要な、SCC によって提供される一部の機能が欠けています。したがって、OpenShift Container Platform は引き続き PSA と SCC の両方をサポートしています。

16.4.1.7. CNF デプロイメントの主な領域

クラウドネイティブネットワーク機能 (CNF) のデプロイメントには、次の主な領域があります。

コア

CNF が初めてデプロイされたのは、ワイヤレスネットワークのコア領域でした。CNF をコアにデプロイするということは、通常、中央オフィスまたはデータセンターにサーバーのラックを配置することを意味します。これらのサーバーは、インターネットと無線アクセスネットワーク (RAN) の両方に接続されます。ただし、多くの場合、複数のセキュリティーファイアウォールの背後に配置され、インターネットから完全に遮断されていることもあります。このタイプのセットアップは、オフラインクラスターまたは非接続クラスターと呼ばれます。

RAN

CNF がコアネットワークで問題なくテストされ、効果的であることが確認されたところで、CNF を無線アクセスネットワーク (RAN) にデプロイすることが検討されました。RAN に CNF を導入するには、多数のサーバー (大規模なデプロイメントでは最大 100,000 台) が必要です。これらのサーバーは、携帯電話基地局の近くに設置されます。通常はシングルノード OpenShift クラスターとして実行され、高いスケーラビリティーを必要とします。

16.4.1.8. 通信事業者固有のインフラストラクチャー

ハードウェア要件

通信事業者のネットワークでは、クラスターは主にベアメタルハードウェア上に構築されます。その場合、仮想マシンを使用せずに、オペレーティングシステム (op-system-first) を物理マシンに直接インストールします。これにより、ネットワーク接続の複雑さが軽減され、レイテンシーが最小限に抑えられ、アプリケーションの CPU 使用率が最適化されます。

ネットワーク要件

通信事業者のネットワークでは、標準的な IT ネットワークよりもはるかに大きな帯域幅が必要です。通信事業者のネットワークでは、大量のデータスループットを処理するために、通常、デュアルポート 25 GB 接続または 100 GB ネットワークインターフェイスカード (NIC) が使用されます。セキュリティーは極めて重要であり、機密性の高い個人データを保護するには、暗号化された接続とセキュアなエンドポイントが必要です。

16.4.1.9. ライフサイクル管理

セキュリティーを確保するには、アップグレードが重要です。脆弱性が発見された場合、その脆弱性は最新の z-stream リリースで修正されます。その後、この修正は下位の各 y-stream リリースに遡って導入され、サポート対象のすべてのバージョンにパッチが適用されます。パッチはサポート対象外になったリリースには適用されません。したがって、OpenShift Container Platform クラスターを定期的にアップグレードして、サポート対象のリリースを維持し、脆弱性から保護された状態を維持することが重要です。

ライフサイクル管理とアップグレードの詳細は、「通信事業者コア CNF クラスターのアップグレード」を参照してください。

関連情報

通信事業者コア CNF クラスターのアップグレード

16.4.1.10. ネットワーク機能から CNF への進化

ネットワーク機能 (NF) は、独立して動作する専用のハードウェアデバイスである物理ネットワーク機能 (PNF) として初まりました。PNF は仮想ネットワーク機能 (VNF) へと徐々に進化しました。VNF は、PNF の機能を仮想化する一方で、CPU、メモリー、ストレージ、ネットワークなどのリソースを

制御します。

テクノロジーがさらに進歩するにつれて、VNF はクラウドネイティブネットワーク機能 (CNF) に移行しました。CNF は軽量でセキュアかつスケーラブルなコンテナー内で実行されます。セキュリティーとパフォーマンスを強化するために、非 root での実行や最小限のホスト干渉など、厳格な制限が適用されます。

PNF は、干渉を受けることなく独立して動作するように、無制限の root アクセス権を持っていました。VNF への移行に伴い、リソースの使用が制御されるようになりましたが、プロセスは仮想マシン内で引き続き root として実行できました。一方、CNF は root アクセスを制限し、コンテナーの機能を制限して、他のコンテナーやホストオペレーティングシステムへの干渉を防ぎます。

CNF への移行における主な課題としては、次のものがあります。

- モノリシックなネットワーク機能を、コンテナー化されたより小さなプロセスに分割する。
- 非 root での実行や分離などのクラウドネイティブの原則に準拠しながら、通信業界グレードのパフォーマンスと信頼性を維持する。

16.4.2. ホストのセキュリティー

16.4.2.1. Red Hat Enterprise Linux CoreOS (RHCOS)

Red Hat Enterprise Linux CoreOS (RHCOS) は、いくつかの重要な点で Red Hat Enterprise Linux (RHEL) と異なります。詳細は、「RHCOS について」を参照してください。

通信事業の観点から見ると、大きな違いは、Machine Config Operator を通じて更新される **rpm-ostree** の制御です。

RHCOS は、OpenShift Container Platform の Pod に使用されるものと同じイミュータブルな設計に準拠しています。これにより、クラスター全体でオペレーティングシステムの一貫性が維持されます。RHCOS アーキテクチャーの詳細は、「Red Hat Enterprise Linux CoreOS (RHCOS)」を参照してください。

セキュリティーを維持しながらホストを効果的に管理するには、可能な限り直接アクセスを避けてください。代わりに、次の方法を使用してホストを管理できます。

- デバッグ Pod
- 直接 SSH 接続
- コンソールアクセス

次の RHCOS セキュリティーメカニズムを確認してください。いずれもホストのセキュリティーを維持するために不可欠なものです。

Linux 名前空間

プロセスとリソースの分離を実現します。各コンテナーは、独自の名前空間内にプロセスとファイルを保持します。ユーザーがコンテナーの名前空間から脱出すると、ユーザーがホストオペレーティングシステムにアクセスできるようになり、セキュリティーが侵害される可能性があります。

Security-Enhanced Linux (SELinux)

必須アクセス制御を適用し、プロセスによるファイルとディレクトリーへのアクセスを制限します。プロセスが制限を破ろうとした場合にファイルへの不正アクセスを防止することにより、セキュリティー層をさらに追加します。

SELinux は、明示的に許可されない限りすべてを拒否するというセキュリティーポリシーに従います。プロセスが許可なくファイルを変更またはアクセスしようとすると、SELinux はアクセスを拒否します。詳細は、SELinux の概要 を参照してください。

Linux ケイパビリティー

プロセスに特定の権限を細かいレベルで割り当てることで、フル root 権限の必要性を最小限に抑えます。詳細は、「Linux ケイパビリティー」を参照してください。

コントロールグループ (cgroup)

プロセスやコンテナーの CPU やメモリーなどのシステムリソースを割り当てて管理し、効率的な使用を実現します。OpenShift Container Platform 4.16 以降、cgroups には 2 つのバージョンがあります。デフォルトでは cgroup v2 が設定されます。

CRI-O

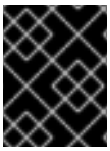
セキュリティー境界を適用し、コンテナーのワークロードを管理する軽量のコンテナーランタイムとして機能します。

関連情報

- RHCOS について
- Red Hat Enterprise Linux CoreOS (RHCOS)
- Linux ケイパビリティー

16.4.2.2. コマンドラインからのホストへのアクセス

ホストの変更やアクセスすべきでない Pod へのアクセスを避けるために、ホストへの直接アクセスを制限する必要があります。ホストに直接アクセスする必要があるユーザーには、LDAP を使用した SSSD などの外部オーセンティケーターを使用してアクセスを管理することを推奨します。これにより、Machine Config Operator を通じてクラスター全体の一貫性を維持できます。



重要

OpenShift Container Platform クラスターサーバーで root ID への直接アクセスを設定しないでください。

クラスター内のノードに接続するには、次の方法を使用できます。

デバッグ Pod の使用

これは推奨されるノードへのアクセス方法です。ノードをデバッグまたは接続するには、次のコマンドを実行します。

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

ノードに接続した後、次のコマンドを実行してルートファイルシステムにアクセスします。

```
# chroot /host
```

これにより、ノード上のデバッグ Pod 内で root アクセスが可能になります。詳細は、「root アクセスでのデバッグ Pod の起動」を参照してください。

直接 SSH 接続

root ユーザーの使用は避けてください。代わりに、コアユーザー ID (または独自の ID) を使用してください。SSH を使用してノードに接続するには、次のコマンドを実行します。

```
$ ssh core@<worker_node_name>
```

重要

コアユーザー ID には、最初にクラスター内で **sudo** 権限が付与されます。

SSH を使用してノードに接続できない場合は、How to connect to OpenShift Container Platform 4.x Cluster nodes using SSH bastion pod を参照して、SSH 鍵をコアユーザーに追加してください。

SSH を使用してノードに接続した後、次のコマンドを実行して root シェルにアクセスします。

```
$ sudo -i
```

コンソールアクセス

コンソールがセキュアであることを確認してください。root ID による直接ログインを許可しないでください。代わりに個別の ID を使用します。

注記

コンソールアクセスの保護に関する組織のベストプラクティスに従ってください。

関連情報

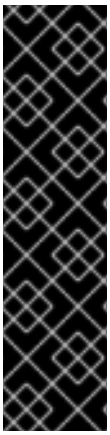
- root アクセスでのデバッグ Pod の起動

16.4.2.3. Linux ケイパビリティー

Linux ケイパビリティーは、プロセスがホストシステム上で実行できるアクションを定義するものです。デフォルトでは、セキュリティー対策が適用されない限り、いくつかのケイパビリティーが Pod に付与されます。これらのデフォルトのケイパビリティーは次のとおりです。

- **CHOWN**
- **DAC_OVERRIDE**
- **FSETID**
- **FOWNER**
- **SETGID**
- **SETUID**
- **SETPCAP**
- **NET_BIND_SERVICE**
- **KILL**

Security Context Constraints (SCC) を設定することで、Pod に付与されるケイパビリティーを変更できます。



重要

次のケイパビリティーを Pod に割り当てることはできません。

- **SYS_ADMIN**: 昇格された権限を付与する強力なケイパビリティー。このケイパビリティーを許可すると、セキュリティー境界が破られ、重大なセキュリティーリスクが生じる可能性があります。
- **NET_ADMIN**: SR-IOV ポートなどのネットワークを制御できます。ただし、最新のセットアップでは代替ソリューションに置き換えることができます。

Linux ケイパビリティーの詳細は、Linux capabilities man ページを参照してください。

16.4.3. Security Context Constraints

RBAC リソースがユーザーアクセスを制御するのと同様の方法で、管理者は Security Context Constraints (SCC) を使用して Pod の権限を制御できます。この権限によって、Pod が実行できるアクションとアクセスできるリソースが決まります。SCC を使用すると、Pod が実行する必要がある一連の条件を定義できます。

Security Context Constraints を使用すると、管理者は次のセキュリティー制約を制御できます。

- Pod が **allowPrivilegedContainer** フラグが付いた特権付きコンテナーを実行できるかどうか
- Pod が **allowPrivilegeEscalation** フラグで制約されているかどうか
- コンテナーが要求できる機能
- ホストディレクトリーのボリュームとしての使用
- コンテナーの SELinux コンテキスト
- コンテナーのユーザー ID
- ホストの namespace とネットワークの使用
- Pod ボリュームを所有する **FSGroup** の割り当て
- 許可される補助グループの設定
- コンテナーが root ファイルシステムへの書き込みアクセスを必要とするかどうか
- ボリュームタイプの使用
- 許可される **seccomp** プロファイルの設定

デフォルトの SCC は、インストール中、および一部の Operator またはその他のコンポーネントをインストールするときに作成されます。クラスター管理者は、OpenShift CLI (**oc**) を使用して独自の SCC を作成することもできます。

デフォルトの Security Context Constraints の詳細は、デフォルトの Security Context Constraints を参照してください。

重要

デフォルトの SCC は変更しないでください。デフォルトの SCC をカスタマイズすると、プラットフォームの Pod をデプロイ時または OpenShift Container Platform のアップグレード時に問題が発生する可能性があります。さらに、一部のクラスターのアップグレード中にデフォルトの SCC 値がデフォルトにリセットされ、それらの SCC に対するすべてのカスタマイズが破棄されます。

デフォルトの SCC を変更する代わりに、必要に応じて独自の SCC を作成および変更します。詳細な手順は、Security Context Constraints の作成 を参照してください。

次の基本的な SCC を使用できます。

- **restricted**
- **restricted-v2**

restricted-v2 SCC は、新規インストールによって提供される最も制限の厳しい SCC であり、認証されたユーザーにデフォルトで使用されます。これは、Pod Security Admission (PSA) の制限と同等であり、オリジナルの **restricted** SCC よりも制限が厳しいため、セキュリティーが強化されます。また、複数のリリースにわたってオリジナルの SCC から v2 に移行する際に有用です。最終的には、オリジナルの SCC は非推奨になります。したがって、**restricted-v2** SCC を使用することを推奨します。

次のコマンドを実行すると、**restricted-v2** SCC を調べることができます。

```
$ oc describe scc restricted-v2
```

出力例

```
Name:                                   restricted-v2
Priority:                               <none>
Access:
  Users:                                <none>
  Groups:                               <none>
Settings:
  Allow Privileged:                     false
  Allow Privilege Escalation:           false
  Default Add Capabilities:             <none>
  Required Drop Capabilities:           ALL
  Allowed Capabilities:                 NET_BIND_SERVICE
  Allowed Seccomp Profiles:             runtime/default
  Allowed Volume Types:
configMap,downwardAPI,emptyDir,ephemeral,persistentVolumeClaim,projected,secret
  Allowed Flexvolumes:                  <all>
  Allowed Unsafe Sysctls:               <none>
  Forbidden Sysctls:                    <none>
  Allow Host Network:                   false
  Allow Host Ports:                     false
  Allow Host PID:                       false
  Allow Host IPC:                       false
  Read Only Root Filesystem:            false
  Run As User Strategy: MustRunAsRange
    UID:                                <none>
    UID Range Min:                      <none>
    UID Range Max:                      <none>
```

```
SELinux Context Strategy: MustRunAs
  User:                               <none>
  Role:                               <none>
  Type:                               <none>
  Level:                              <none>
FSGroup Strategy: MustRunAs
  Ranges:                               <none>
Supplemental Groups Strategy: RunAsAny
  Ranges:                               <none>
```

restricted-v2 SCC は、明示的に許可されているもの以外はすべて明示的に拒否します。次の設定により、許可されるケイパビリティーとセキュリティー制限が定義されます。

- Default add capabilities: **<none>** に設定されています。つまり、デフォルトでは Pod にケイパビリティーが追加されません。
- Required drop capabilities: **ALL** に設定されています。これにより、Pod のデフォルトの Linux ケイパビリティーがすべて削除されます。
- Allowed capabilities: **NET_BIND_SERVICE**。Pod はこのケイパビリティーを要求できますが、デフォルトでは追加されません。
- Allowed **seccomp** profiles: **runtime/default**。

詳細は、Security Context Constraints の管理 を参照してください。