



OpenShift Container Platform 4.15

エッジコンピューティング

ネットワークエッジでの OpenShift Container Platform クラスターの設定およびデプロイ

OpenShift Container Platform 4.15 エッジコンピューティング

ネットワークエッジでの OpenShift Container Platform クラスターの設定およびデプロイ

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このドキュメントでは、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、ネットワークの遠端にあるサイトをプロビジョニングおよび管理する方法について説明します。

目次

| | |
|--|-----------|
| 第1章 ネットワークファー遠端の課題 | 5 |
| 1.1. ネットワークファーエッジの課題を克服する | 5 |
| 1.2. GITOPS ZTP を使用したネットワーク遠端でのクラスタープロビジョニング | 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 ZTP 用のハブクラスターの準備 | 11 |
| 2.1. TELCO RAN DU 4.15 の検証済みソフトウェアコンポーネント | 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. 切断された環境での 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. 支援サービスの有効化 | 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. ARG OCD 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 | 21 |
| 第3章 GITOPS ZTP の更新 | 26 |
| 3.1. GITOPS ZTP 更新プロセスの概要 | 26 |
| 3.2. アップグレードの準備 | 26 |
| 3.3. 既存クラスターのラベル付け | 27 |
| 3.4. 既存の GITOPS ZTP アプリケーションの停止 | 27 |
| 3.5. GIT リポジトリーに必要な変更 | 28 |
| 3.6. 新規 GITOPS ZTP アプリケーションのインストール | 30 |
| 3.7. GITOPS ZTP 設定の変更のロールアウト | 30 |
| 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 | 31 |
| 4.1. GITOPS ZTP および TOPOLOGY AWARE LIFECYCLE MANAGER | 31 |
| 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 | 32 |
| 4.3. マネージドベアメタルホストシークレットの作成 | 33 |
| 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 | 34 |
| 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 | 36 |
| 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 | 44 |
| 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 | 45 |
| 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 | 46 |
| 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 | 46 |
| 4.10. GITOPS ZTP パイプラインからの古いコンテンツの削除 | 47 |
| 4.11. GITOPS ZTP パイプラインの破棄 | 47 |
| 第5章 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 | 49 |
| 5.1. POLICYGENTEMPLATE CRD について | 49 |
| 5.2. POLICYGENTEMPLATE CR をカスタマイズする際の推奨事項 | 52 |
| 5.3. RAN デプロイメントの POLICYGENTEMPLATE CR | 53 |
| 5.4. POLICYGENTEMPLATE CR を使用したマネージドクラスターのカスタマイズ | 54 |
| 5.5. 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 | 56 |
| 5.6. 設定ポリシー CR の生成の検証 | 57 |
| 5.7. ポリシー調整の再開 | 59 |
| 5.8. 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 | 60 |
| 5.9. GITOPS ZTP インストール完了の表示 | 62 |
| 第6章 ZTP を使用した単一ノード OPENSIFT クラスターの手動インストール | 64 |
| 6.1. GITOPS ZTP インストール CR と設定 CR の手動生成 | 64 |

| | |
|--|------------|
| 6.2. マネージドベアメタルホストシークレットの作成 | 69 |
| 6.3. 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 | 70 |
| 6.4. 単一のマネージドクラスターのインストール | 72 |
| 6.5. 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 | 73 |
| 6.6. マネージドクラスターのトラブルシューティング | 74 |
| 6.7. 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 | 75 |
| 第7章 VDU アプリケーションのワークロードに推奨される単一ノードの OPENSIFT クラスター設定 | 77 |
| 7.1. OPENSIFT CONTAINER PLATFORM で低レイテンシーのアプリケーションを実行する | 77 |
| 7.2. 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 | 77 |
| 7.3. 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 | 78 |
| 7.4. マネージドクラスターネットワークの接続の前提条件 | 79 |
| 7.5. GITOPS ZTP を使用した単一ノードの OPENSIFT でのワークロードの分割 | 79 |
| 7.6. 推奨されるクラスターインストールマニフェスト | 80 |
| 7.7. 推奨されるインストール後のクラスター設定 | 95 |
| 第8章 VDU アプリケーションワークロードの単一ノード OPENSIFT クラスターチューニングの検証 | 120 |
| 8.1. VDU クラスターホストの推奨ファームウェア設定 | 120 |
| 8.2. 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 | 122 |
| 8.3. 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 | 125 |
| 第9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 | 136 |
| 9.1. 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 | 136 |
| 9.2. 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 | 137 |
| 第10章 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 | 140 |
| 10.1. 追加の変更のクラスターへのデプロイ | 140 |
| 10.2. POLICYGENTEMPLATE CR を使用して、ソース CR の内容を上書きする。 | 140 |
| 10.3. GITOPS ZTP パイプラインへのカスタムコンテンツの追加 | 143 |
| 10.4. 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 | 145 |
| 10.5. 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 | 147 |
| 10.6. POLICYGENTEMPLATES CR を使用して電源状態を設定する | 148 |
| 10.7. POLICYGENTEMPLATE CR を使用した LVM ストレージの設定 | 152 |
| 10.8. POLICYGENTEMPLATE CR を使用した PTP イベントの設定 | 154 |
| 10.9. POLICYGENTEMPLATE CR を使用したベアメタルイベントの設定 | 158 |
| 10.10. 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 | 161 |
| 10.11. POLICYGENTEMPLATE CR でのハブテンプレートの使用 | 166 |
| 第11章 TOPOLOGY AWARE LIFECYCLE MANAGER を使用したマネージドクラスターの更新 | 173 |
| 11.1. TOPOLOGY AWARE LIFECYCLE MANAGER の設定について | 173 |
| 11.2. TOPOLOGY AWARE LIFECYCLE MANAGER で使用される管理ポリシー | 173 |
| 11.3. WEB コンソールを使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール | 174 |
| 11.4. CLI を使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール | 174 |
| 11.5. CLUSTERGROUPUPGRADE CR | 176 |
| 11.6. マネージドクラスターでのポリシーの更新 | 192 |
| 11.7. アップグレード前のクラスターリソースのバックアップの作成 | 201 |
| 11.8. コンテナイメージ事前キャッシュ機能の使用 | 207 |
| 11.9. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング | 211 |
| 第12章 TOPOLOGY AWARE LIFECYCLE MANAGER 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 ... | 221 |
| 12.1. 切断された環境でのクラスターの更新 | 221 |
| 12.2. GITOPS ZTP 用に自動作成された CLUSTERGROUPUPGRADE CR について | 241 |

| | |
|--|------------|
| 第13章 GITOPS ZTP を使用した単一ノードの OPENSIFT クラスターの拡張 | 244 |
| 13.1. ワーカーノードへのプロファイルの適用 | 244 |
| 13.2. (オプション) PTP および SR-IOV デモンセクターの互換性の確保 | 245 |
| 13.3. PTP および SR-IOV ノードセクターの互換性 | 246 |
| 13.4. 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 | 246 |
| 13.5. GITOPS ZTP を使用して単一ノードの OPENSIFT クラスターにワーカーノードを追加する | 249 |
| 第14章 単一ノードの OPENSIFT デプロイメント用のイメージの事前キャッシュ | 253 |
| 14.1. FACTORY-PRECACHEING-CLI ツールの入手 | 253 |
| 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 | 254 |
| 14.3. ディスクのパーティション設定 | 255 |
| 14.4. イメージのダウンロード | 259 |
| 14.5. GITOPS ZTP でのイメージの事前キャッシュ | 268 |
| 14.6. トラブルシューティング | 272 |

第1章 ネットワークファアー遠端の課題

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

1.1. ネットワークファアーエッジの課題を克服する

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

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

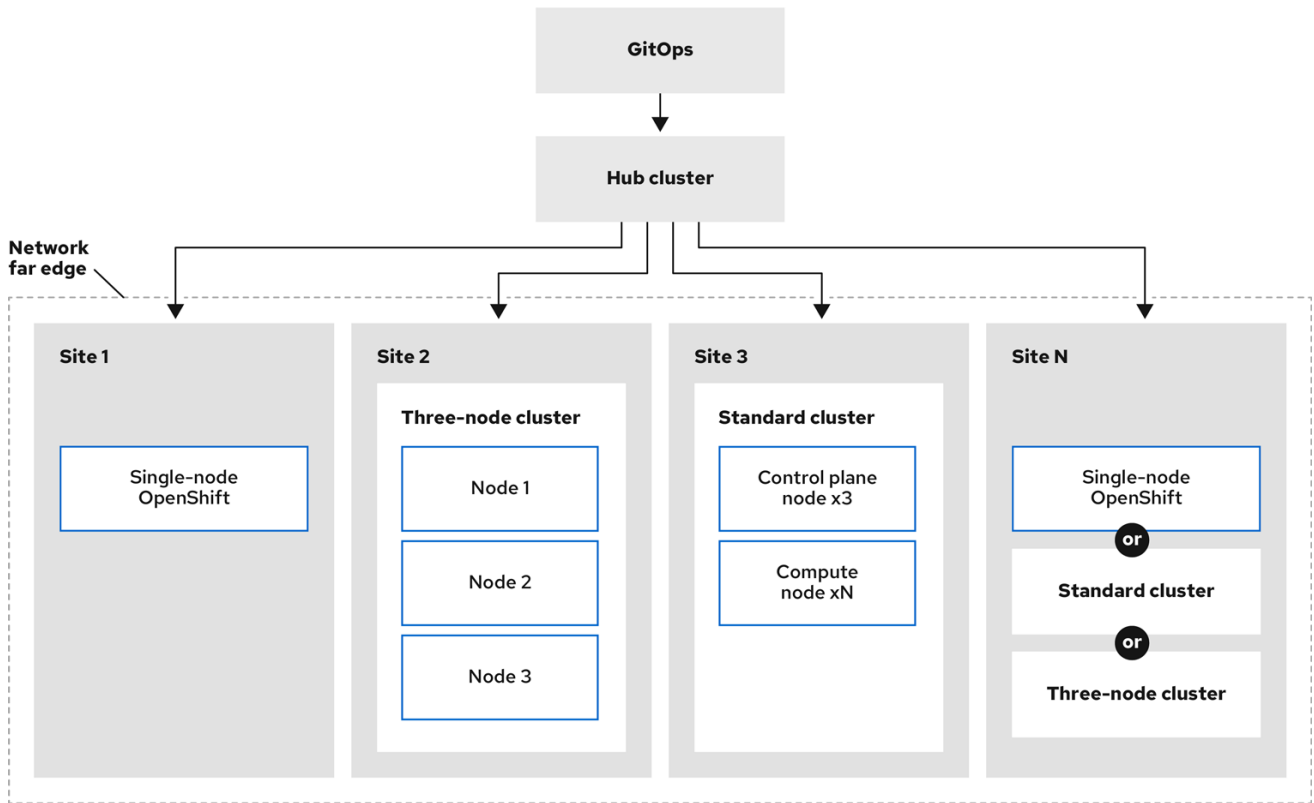
GitOps Zero Touch Provisioning (ZTP) と **GitOps** は、ベアメタル機器の宣言的なサイト定義と設定を使用してリモートエッジサイトを大規模にプロビジョニングできるようにすることで、これらの課題を解決します。テンプレートまたはオーバーレイ設定は、CNF ワークロードに必要な OpenShift Container Platform 機能をインストールします。インストールとアップグレードの全ライフサイクルは、GitOps ZTP パイプラインを通じて処理されます。

GitOps ZTP は、インフラストラクチャーのデプロイメントに GitOps を使用します。GitOps では、Git リポジトリ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 リポジトリを使用してインフラストラクチャーのデプロイメントを推進します。

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

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

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



217_OpenShift_1022

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

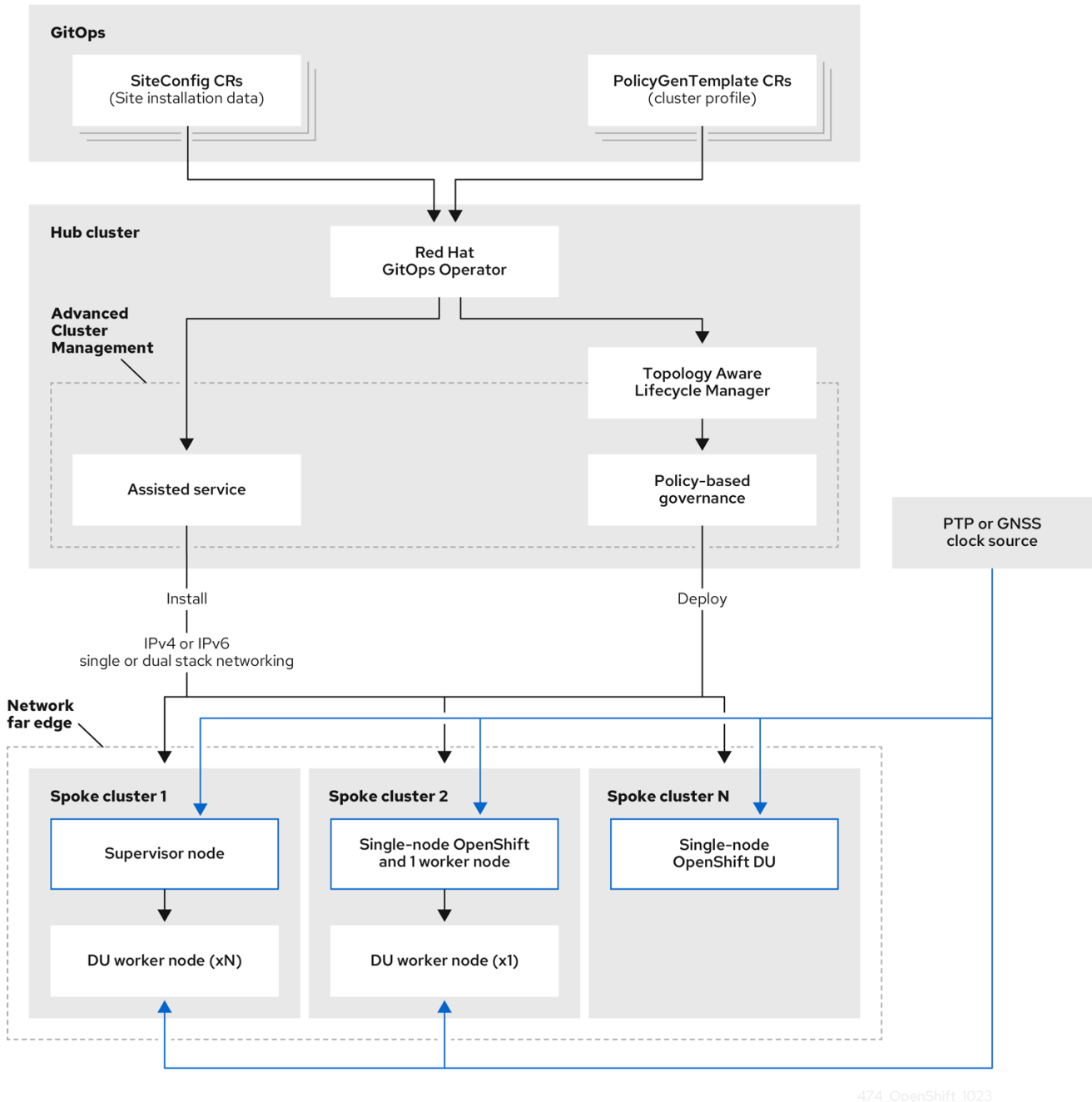
Red Hat Advanced Cluster Management (RHACM) は、単一のハブクラスターが多数のスポーククラスターを管理するハブアンドスポークアーキテクチャーでクラスターを管理します。RHACM を実行するハブクラスターは、GitOps Zero Touch Provisioning (ZTP) と、RHACM のインストール時にデプロイメントされるアシストサービスを使用して、マネージドクラスターのプロビジョニングおよびデプロイメントを実行します。

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

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

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

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



474_OpenShift_1023

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。

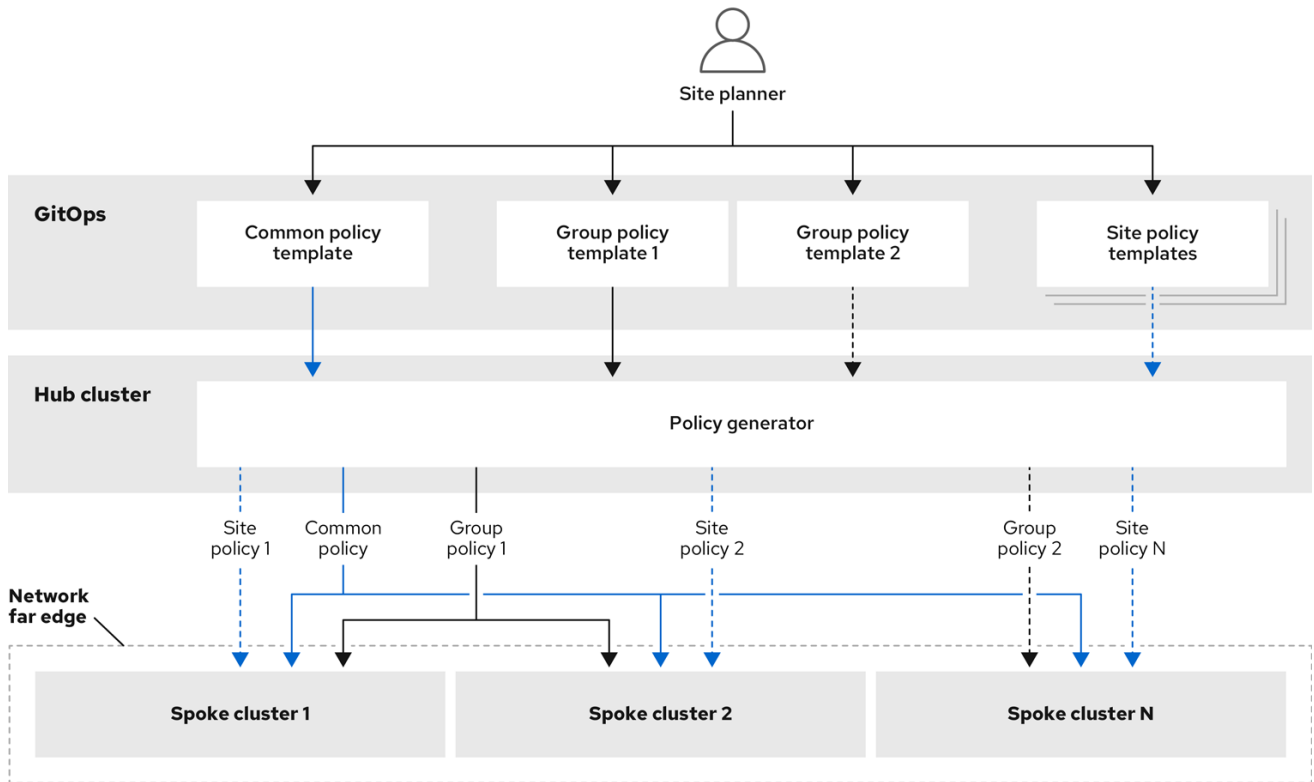
ポリシージェネレーターまたは **PolicyGen** は、簡潔なテンプレートから RHACM ポリシーを作成できるようにする GitOps Operator のプラグインです。このツールは、複数の CR を 1 つのポリシーに組み合わせることができ、フリート内のクラスターのさまざまなサブセットに適用される複数のポリシーを生成できます。

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。



217_OpenShift_1022

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。
- クラスターバリエーション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

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

- **ztp-site-generate** コンテナイメージから参照 **SiteConfig** および **PolicyGenTemplate** CR を抽出する方法の詳細は、[ZTP Git リポジトリの準備](#) を参照してください。

第2章 ZTP 用のハブクラスターの準備

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

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

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

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

| コンポーネント | ソフトウェアバージョン |
|--------------------------|-------------|
| マネージドクラスターのバージョン | 4.15 |
| Cluster Logging Operator | 5.8 |
| Local Storage Operator | 4.15 |
| PTP Operator | 4.15 |
| SRIOV Operator | 4.15 |
| Node Tuning Operator | 4.15 |
| Logging Operator | 4.15 |
| SRIOV-FEC Operator | 2.8 |

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 | 4.15 |
| GitOps ZTP プラグイン | 4.15 |
| Red Hat Advanced Cluster Management (RHACM) | 2.9, 2.10 |
| Red Hat OpenShift GitOps | 1.11 |
| Topology Aware Lifecycle Manager (TALM) | 4.15 |

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める重要な要素です。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める重要な要素です。

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に制限される可能性があります。

RHACM のバージョンと設定

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

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

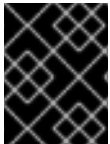


重要

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

表2.3 代表的な 3 ノードハブクラスターマシンの仕様

| 要件 | 説明 |
|---|---|
| OpenShift Container Platform | バージョン 4.13 |
| RHACM | バージョン 2.7 |
| Topology Aware Lifecycle Manager (TALM) | バージョン 4.13 |
| サーバーハードウェア | Dell PowerEdge R650 ラックサーバー 3 台 |
| NVMe ハードディスク | <ul style="list-style-type: none"> ● <code>/var/lib/etcd</code> 用の 50 GB ディスク ● <code>/var/lib/containers</code> 用の 2.9 TB ディスク |
| SSD ハードディスク | <ul style="list-style-type: none"> ● 1つの SSD を、15 のシンプロビジョニングされた 200 GB の論理ボリュームに分割して PV CR としてプロビジョニング。 ● 非常に大規模な PV リソースとして機能する 1つの SSD |
| 適用された DU プロファイルポリシーの数 | 5 |



重要

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

表2.4 模擬ラボ環境のネットワーク仕様

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

関連情報

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

2.3. 切断された環境での GITOPS ZTP のインストール

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

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- クラスタで使用するために、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> ❶
```

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

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

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

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

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

- b. 必要なイメージをダウンロードします。

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.15/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.15/${OCP_VERSION}/${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-4.15.1-x86_64-live.x86_64.iso
rhcos-4.15.1-x86_64-live.x86_64.iso- 11%[====> ] 10.01M 4.71MB/s
```

関連情報

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

2.5. 支援サービスの有効化

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

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. 以下のコマンドを実行して、**AgentServiceConfig** CR を更新します。

```
$ oc edit AgentServiceConfig
```

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

```
- cpuArchitecture: x86_64
  openshiftVersion: "4.15"
  rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
  url: https://<mirror-registry>/<path>/rhcos-live.x86_64.iso
```

ここでは、以下のようになります。

<host>

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

<path>

ターゲットミラーレジストリー上のイメージへのパスです。

エディターを保存して終了し、変更を適用します。

2.6. 切断されたミラーレジストリーを使用するためのハブクラスターの設定

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

前提条件

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



警告

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

手順

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

```

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

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

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

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

- ❶ **ConfigMap** namespace は **multicluster-engine** に設定する必要があります。
- ❷ 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。
- ❸ 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イメージをフェッチします。
- ❹ ミラーレジストリーの URL。ミラーレジストリーを設定する場合は、`oc adm release Mirror` コマンドを実行して、`imageContentSources` セクションの URL を使用する必要があります。詳細は、[OpenShift Container Platform イメージリポジトリーのミラーリン](#)

グセクションを参照してください。

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

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

出力例

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

- 1 **ConfigMap** namespace と一致するように、**AgentServiceConfig** namespace を **multicluster-engine** に設定します。
- 2 関連する **ConfigMap** CR で指定された定義と一致するように、**mirrorRegistryRef.name** を設定します。
- 3 **httpd** サーバーでホストされる ISO の URL を設定します。



重要

クラスタ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

- ハブクラスターをインストールして設定し、ハブクラスターに Red Hat Advanced Cluster Management (RHACM) をインストールしている。
- OpenShift Container Platform CLI (oc) をインストールしている。
- **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** など終わっている必要があります。 <https://repo.example.com/repo.git> とクレデンシャルを指定します。
 - **certificates**: 必要に応じて、リポジトリーのパブリック証明書を追加します。

- c. 2つの ArgoCD アプリケーション、**out/argocd/deployment/clusters-app.yaml** と **out/argocd/deployment/policies-app.yaml** を、Git リポジトリに基づいて修正します。
- Git リポジトリを参照するように 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 リポジトリブランチを示します。
 - **path** は、それぞれ **SiteConfig** CR および **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ために、以前に **out/argocd/deployment/** ディレクトリに展開したパッチファイルを使用して、ハブクラスター内の ArgoCD インスタンスにパッチを適用します。以下のコマンドを実行します。

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

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

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

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

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

2.9. GITOPS ZTP サイト設定リポジトリの準備

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** CR と **PolicyGenTemplate** CR の個別のパスを持つディレクトリ構造を作成します。



注記

SiteConfig および **PolicyGenTemplate** CR を個別のディレクトリで保持します。**SiteConfig** ディレクトリおよび **PolicyGenTemplate** ディレクトリには、そのディレクトリ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。

- 以下のコマンドを使用して **ztp-site-generate** コンテナイメージから **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:v4.15
```

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.15 extract /home/ztp --tar | tar x -C ./out
```

- out** ディレクトリーに以下のサブディレクトリーが含まれていることを確認します。
 - out/extra-manifest** には、**SiteConfig** が追加の manifest **configMap** の生成に使用するソース CR ファイルが含まれます。
 - out/source-crs** には、**PolicyGenTemplate** が Red Hat Advanced Cluster Management (RHACM) ポリシーを生成するために使用するソース CR ファイルが含まれています。
 - out/argocd/deployment** には、この手順の次のステップで使用するハブクラスターに適用するパッチおよび YAML ファイルが含まれます。
 - out/argocd/example** には、推奨の設定を表す **SiteConfig** ファイルおよび **PolicyGenTemplate** ファイルのサンプルが含まれています。
- out/source-crs** フォルダーとその内容を **PolicyGentemplate** ディレクトリーにコピーします。
- 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/
├── policygentemplates
│   ├── kustomization.yaml
│   └── source-crs/
├── siteconfig
│   ├── extra-manifests
│   └── kustomization.yaml
```

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

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

すべてのクラスタータイプについて、次のことを行う必要があります。

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

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

```
example/
├── policygentemplates
│   ├── common-ranGen.yaml
│   ├── example-sno-site.yaml
│   ├── 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。

2.9.1. 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

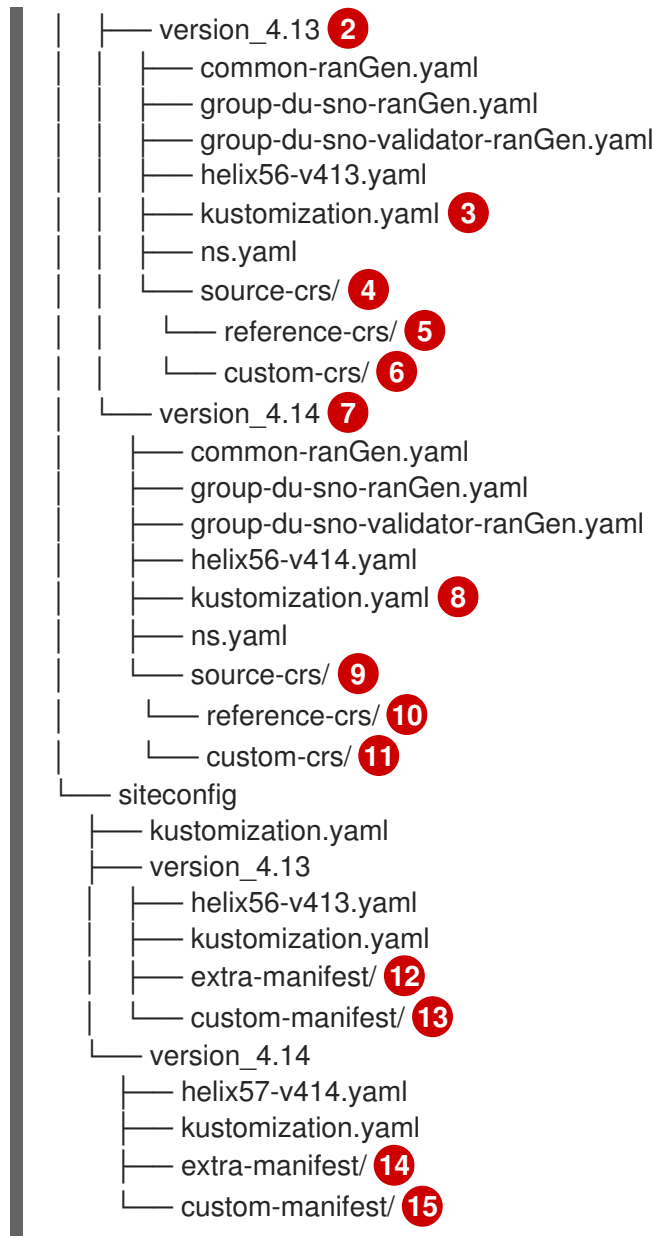
GitOps ZTP を使用して、OpenShift Container Platform のさまざまなバージョンを実行しているマネージドクラスターのソースカスタムリソース (CR) を管理できます。これは、ハブクラスター上で実行している OpenShift Container Platform のバージョンが、マネージドクラスター上で実行しているバージョンから独立している可能性があることを意味します。

手順

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

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

```
├── policygentemplates
│   └── kustomization.yaml ①
```



- 1 最上位の **kustomization** YAML ファイルを作成します。
- 2 7 カスタム **/policygentemplates** 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。

3.1. GITOPS ZTP 更新プロセスの概要

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



注記

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

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

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

3.2. アップグレードの準備

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

手順

1. GitOps ZTP で使用するために Red Hat OpenShift GitOps を設定するために使用されるカスタムリソース (CR) を持つ GitOps ZTP コンテナ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:v4.15 extract /home/ztp --tar | tar x -C ./update
```

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

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

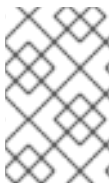


重要

GitOps ZTP を最新バージョンに更新するときは、**update/argocd/deployment** ディレクトリからサイトリポジトリ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 リポジトリに必要な変更

ztp-site-generate コンテナ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、リポジトリ内の既存のコンテンツを更新する必要があります。

- **PolicyGenTemplate** ファイルに必要な変更を加えます。
すべての **PolicyGenTemplate** ファイルは、**ztp** で始まる **Namespace** で作成する必要があります。これにより、GitOps ZTP アプリケーションは、Red Hat Advanced Cluster Management (RHACM) が内部でポリシーを管理する方法と競合することなく、GitOps ZTP によって生成されたポリシー CR を管理できるようになります。
- **kustomization.yaml** ファイルをリポジトリに追加します。
すべての **SiteConfig** および **PolicyGenTemplate** CR は、それぞれのディレクトリーツリーの下にある **kustomization.yaml** ファイルに含める必要があります。以下に例を示します。

```

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

```




注記

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

PolicyGenTemplate kustomization ファイルには、すべての **PolicyGenTemplate** YAML ファイルが **generator** セクションに含まれ、**Namespace** CR が **resource** セクションに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:
- common-ranGen.yaml
- group-du-sno-ranGen.yaml
- site1.yaml
- site2.yaml

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

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

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

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

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



注記

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

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

ネットワーク内のクラスターのタイプに適用可能なリファレンス **SiteConfig** および **PolicyGenTemplate** CR を確認します。これらの例は、GitOps ZTP コンテナから抽出した **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ために、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使用して、ハブクラスター内の ArgoCD インスタンスにパッチを適用します。以下のコマンドを実行します。

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

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

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

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

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

3.7. GITOPS ZTP 設定の変更のロールアウト

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

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

関連情報

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

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

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

4.1. GITOPS ZTP および TOPOLOGY AWARE LIFECYCLE MANAGER

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

Inform ポリシー

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

ClusterGroupUpgrade CR の自動作成

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

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

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

PolicyGenTemplate CR から生成される各ポリシーには、**ztp-deploy-wave** アノテーションが含まれます。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 **PolicyGenTemplate** で上書きできます。ソース CR の wave アノテーションは、ポリシーの wave アノテーションを判別し、設定するために使用されます。このアノテーションは、実行時に生成されるポリシーに含まれるビルドされる各 CR から削除されます。

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

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

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

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

フェーズラベル

ClusterGroupUpgrade CR は自動的に作成され、そこには GitOps ZTP プロセスの開始時と終了時に **ManagedCluster** CR をラベルでアノテートするディレクティブが含まれています。

インストール後に GitOps ZTP 設定が開始されると、**ManagedCluster** に **ztp-running** ラベルが適用されます。すべてのポリシーがクラスターに修復され、完全に準拠されると、TALM は **ztp-running** ラベルを削除し、**ztp-done** ラベルを適用します。

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

リンクされた CR

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

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

Red Hat Advanced Cluster Management (RHACM) は、GitOps Zero Touch Provisioning (ZTP) を使用して、単一ノードの OpenShift Container Platform クラスター、3 ノードのクラスター、および標準ク

クラスターをデプロイします。サイト設定データは、Git リポジトリで OpenShift Container Platform カスタムリソース (CR) として管理します。GitOps ZTP は、宣言的な GitOps アプローチを使用して、一度開発すればどこにでもデプロイできるモデルを使用して、マネージドクラスターをデプロイします。

クラスターのデプロイメントには、以下が含まれます。

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

マネージドサイトのインストールプロセスの概要

マネージドサイトのカスタムリソース (CR) をハブクラスターに適用すると、次のアクションが自動的に実行されます。

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

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



重要

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

4.3. マネージドベアメタルホストシークレットの作成

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



注記

シークレットは、**SiteConfig** CR から名前参照されます。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 ❶
data: ❷
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno ❸
data:
  .dockerconfigjson: <pull_secret> ❹
type: kubernetes.io/dockerconfigjson

```

- ❶ 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
- ❷ **password** と **username** の Base64 エンコード値
- ❸ 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
- ❹ Base64 でエンコードされたプルシークレット

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

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

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



注記

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

前提条件

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

手順

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



注記

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

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "1"
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
spec:
  clusterRef:
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
  kernelArguments:
    - operation: append ①
      value: audit=0 ②
    - operation: append
      value: trace=1
  sshAuthorizedKey: "{{ .Site.SshPublicKey }}"
  proxy: "{{ .Cluster.ProxySettings }}"
  pullSecretRef:
    name: "{{ .Site.PullSecretRef.Name }}"
  ignitionConfigOverride: "{{ .Cluster.IgnitionConfigOverride }}"
  nmStateConfigLabelSelector:
    matchLabels:
      nmstate-label: "{{ .Cluster.ClusterName }}"
  additionalNTPSources: "{{ .Cluster.AdditionalNTPSources }}"
```

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

2. **InfraEnv-example.yaml** CR を、Git リポジトリ内の **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 リポジトリ内の **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なら、ビルドパイプラインは Git リポジトリ内のカスタム **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 リポジトリを作成しています。リポジトリはハブクラスターからアクセスできる必要があります。ArgoCD アプリケーションのソースリポジトリ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の準備」を参照してください。



注記

ソースリポジトリを作成するときは、**ztp-site-generate** コンテナから抽出した **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名前空間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、名前が **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/** フォルダから CR の適切な例を選択します。フォルダには、単一ノード、3 ノード、標準クラスターのサンプルファイルが含まれます。
 - **example-sno.yaml**
 - **example-3node.yaml**

- **example-standard.yaml**

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

シングルノード OpenShift SiteConfig CR の例

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created
under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
    - clusterName: "example-sno"
      networkType: "OVNKubernetes"
      # installConfigOverrides is a generic way of passing install-config
      # parameters through the siteConfig. The 'capabilities' field configures
      # the composable openshift feature. In this 'capabilities' setting, we
      # remove all but the marketplace component from the optional set of
      # components.
      # Notes:
      # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
      installConfigOverrides: "{\"capabilities\":{\"baselineCapabilitySet\": \"None\",
      \"additionalEnabledCapabilities\": [ \"marketplace\", \"NodeTuning\" ] }}"
      # 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: "4.14"
        # 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:
  wwn: "0x11111000000asd123"
# example of diskPartition below is used for image registry (check
ImageRegistry.md for more details), but it's not limited to this use case
#   diskPartition:
#     - device: /dev/disk/by-id/wwn-0x11111000000asd123 # match
rootDeviceHints
#   partitions:
#     - mount_point: /var/imageregistry
#       size: 102500
#       start: 344844

nodeNetwork:
  interfaces:
    - name: eno1
      macAddress: "AA:BB:CC:DD:EE:11"
  config:
    interfaces:
      - name: eno1
        type: ethernet
        state: up
        ipv4:
          enabled: false
        ipv6:
          enabled: true
        address:

```

```

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

```



注記

BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。

- 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 リポジトリに **sno-extra-manifest/** などのディレクトリを作成し、このディレクトリ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を参照する場合、この参照ディレクトリの 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,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= ❶
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

- ❶ 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. 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 | <p>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。</p> <div style="display: flex; align-items: center;">  <div> <p>注記</p> <p>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。</p> </div> </div> |
| metadata.name | <p>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。</p> |

| SiteConfig CR フィールド | 説明 |
|--|--|
| spec.clusterImageSetNameRef | サイト内のすべてのクラスタのハブクラスタ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 | <p>クラスタ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。</p> <div data-bbox="491 607 596 770" style="float: left; width: 60px; height: 60px; background-color: #ccc; border: 1px solid #000; margin-right: 10px;"></div> <p>重要</p> <p>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。</p> |
| spec.clusters.clusterImageSetNameRef | 個々のクラスタをデプロイするために使用されるクラスタ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 PolicyGenTemplate CR の bindingRules フィールドに対応するクラスタラベルを設定します。たとえば、 policygentemplates/common-ranGen.yaml は common: true が設定されたすべてのクラスタに適用され、 policygentemplates/group-du-sno-ranGen.yaml は group-du-sno: "" が設定されたすべてのクラスタに適用されます。 |
| spec.clusters.crTemplates.KlusterletAddonConfig | オプション: klusterletaddonconfig を、クラスタ用に作成された KlusterletAddonConfigOverride.yaml to override the default `KlusterletAddonConfig` に設定します。 |
| spec.clusters.nodes.hostName | 単一ノードの導入では、単一のホストを定義します。3 ノードのデプロイメントの場合、3 台のホストを定義します。標準のデプロイメントでは、 role: master と、 role: worker で定義される 2 つ以上のホストを持つ 3 つのホストを定義します。 |
| spec.clusters.nodes.nodeLabels | マネージドクラスタ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。 |

| SiteConfig CR フィールド | 説明 |
|---|---|
| spec.clusters.nodes.bmcAddress | <p>ホストへのアクセスに使用する BMC アドレス。すべてのクラスタータイプに適用されます。GitOps ZTP は、Redfish または IPMI プロトコルを使用して iPXE および仮想メディアの起動をサポートします。iPXE ブートを使用するには、RHACM 2.8 以降を使用する必要があります。BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。</p> |
| spec.clusters.nodes.bmcAddress | <p>ホストへのアクセスに使用する BMC アドレス。すべてのクラスタータイプに適用されます。GitOps ZTP は、Redfish または IPMI プロトコルを使用して iPXE および仮想メディアの起動をサポートします。iPXE ブートを使用するには、RHACM 2.8 以降を使用する必要があります。BMC アドレッシングの詳細については、「関連情報」セクションを参照してください。</p> <div data-bbox="491 703 596 837" style="float: left; margin-right: 10px;"> </div> <p>注記</p> <p>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。</p> |
| spec.clusters.nodes.bmcCredentialsName | <p>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。</p> |
| spec.clusters.nodes.bootMode | <p>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。</p> |
| spec.clusters.nodes.rootDeviceHints | <p>導入するデバイスを指定します。再起動後も安定した識別子が推奨されます (例: wwn: <disk_wwn> または deviceName:/dev/disk/by-path/<device_path>)。安定した識別子の詳細なリストは、「ルートデバイスのヒント」セクションを参照してください。</p> |
| spec.clusters.nodes.diskPartition | <p>オプション: 提供されるサンプル diskPartition は、追加のディスクパーティションを設定するために使用されます。</p> |
| spec.clusters.nodes.ignitionConfigOverride | <p>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。</p> |
| spec.clusters.nodes.cpuset | <p>ワークロードのパーティショニング用に、クラスター PerformanceProfile CR spec.cpu.reserved フィールドに設定した値と一致するように cpuset を設定します。</p> |
| spec.clusters.nodes.nodeNetwork | <p>ノードのネットワーク設定を行います。</p> |
| spec.clusters.nodes.nodeNetwork.config.interfaces.ipv6 | <p>ホストの IPv6 アドレスを設定します。静的 IP アドレスを持つ単一ノードの OpenShift クラスターの場合、ノード固有の API と Ingress IP は同じである必要があります。</p> |

SiteConfig CR フィールド 説明

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 ZTP クラスターデプロイメントの完了のシグナリング](#)
- [マネージドベアメタルホストシークレットの作成](#)
- [BMC アドレス指定](#)
- [ルートデバイスヒントについて](#)

4.6. マネージドクラスターのインストールの進行状況の監視

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

前提条件

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

手順

同期が完了すると、インストールは一般的に以下のように行われます。

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

- a. クラスター名をエクスポートします。

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

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

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

- c. クラスターのインストールイベントを取得します。


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

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

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

前提条件

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

手順

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

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

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

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

```
$ oc get managedcluster
```

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

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

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

```
Status:
Conditions:
  Last Transition Time: 2021-11-26T17:21:39Z
  Message:          rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/siteconfigs/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not create extra-manifest ranSite1.extra-manifest3 stat
extra-manifest3: no such file or directory 2021/11/26 17:21:40 Error: could not build the
entire SiteConfig defined by /tmp/kust-plugin-config-913473579: stat extra-manifest3: no
such file or directory Error: failure in plugin configured via /tmp/kust-plugin-config-
913473579; exit status 1: exit status 1
  Type: ComparisonError
```

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

```

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

```

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

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

前提条件

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

手順

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

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

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

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

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

前提条件

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

手順

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

2. 任意: サイトを永続的に削除する場合は、Git リポジトリから **SiteConfig** ファイルおよびサイト固有の **PolicyGenTemplate** ファイルも削除する必要があります。
3. 任意: たとえば、サイトを再デプロイする際にサイトを一時的に削除する場合には、Git リポジトリに **SiteConfig** およびサイト固有の **PolicyGenTemplate** CR を残しておくことができます。

関連情報

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

4.10. GITOPS ZTP パイプラインからの古いコンテンツの削除

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

前提条件

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

手順

1. Git リポジトリから影響を受ける **PolicyGenTemplate** ファイルを削除し、コミットしてリモートリポジトリにプッシュしてください。
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 **PolicyGenTemplate** ファイルを Git リポジトリに再び追加し、リモートリポジトリにコミットし、プッシュします。



注記

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

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

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

4.11. GITOPS ZTP パイプラインの破棄

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

前提条件

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

手順

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

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

3. 変更をコミットして、サイトリポジトリーにプッシュします。

第5章 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

適用されたポリシーのカスタムリソース (CR) は、プロビジョニングするマネージドクラスターを設定します。Red Hat Advanced Cluster Management (RHACM) が **PolicyGenTemplate** CR を使用して、適用されるポリシー CR を生成する方法をカスタマイズできます。

5.1. POLICYGENTEMPLATE CRD について

PolicyGenTemplate カスタムリソース定義 (CRD) は、**PolicyGen** ポリシージェネレーターに、どのカスタムリソース (CR) をクラスター設定に含めるか、CR を生成されたポリシーに結合する方法、およびこれらの CR 内のどのアイテムをオーバーレイコンテンツで更新する必要があるかを伝えます。

次の例は、**ztp-site-generate** 参照コンテナーから抽出された **PolicyGenTemplate** CR (**common-du-ranGen.yaml**) を示しています。**common-du-ranGen.yaml** ファイルは、2つの Red Hat Advanced Cluster Management (RHACM) ポリシーを定義します。ポリシーは、CR 内の **policyName** の一意の値ごとに1つずつ、設定 CR のコレクションを管理します。**common-du-ranGen.yaml** は、単一の配置バインディングと配置ルールを作成して、**bindingRules** セクションにリストされているラベルに基づいてポリシーをクラスターにバインドします。

PolicyGenTemplate CR の例 - common-du-ranGen.yaml

```
---
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "common"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true" ①
  sourceFiles: ②
  - fileName: SriovSubscription.yaml
    policyName: "subscriptions-policy"
  - fileName: SriovSubscriptionNS.yaml
    policyName: "subscriptions-policy"
  - fileName: SriovSubscriptionOperGroup.yaml
    policyName: "subscriptions-policy"
  - fileName: SriovOperatorStatus.yaml
    policyName: "subscriptions-policy"
  - fileName: PtpSubscription.yaml
    policyName: "subscriptions-policy"
  - fileName: PtpSubscriptionNS.yaml
    policyName: "subscriptions-policy"
  - fileName: PtpSubscriptionOperGroup.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: ReduceMonitoringFootprint.yaml
  policyName: "config-policy"
- fileName: OperatorHub.yaml ❸
  policyName: "config-policy"
- fileName: DefaultCatsrc.yaml ❹
  policyName: "config-policy" ❺
  metadata:
    name: redhat-operators
  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

```

- ❶ **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。
- ❷ **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。
- ❸ **OperatorHub.yaml** は、切断されたレジストリーの OperatorHub を設定します。
- ❹ **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。
- ❺ **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-ntp-slave"
  spec:
    profile:
      - name: "slave"
        interface: "ens5f0"
        ptp4IOpts: "-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-ntp-slave** という名前です。**PtpConfigSlave.yaml** で定義された **spec** は、**du-ntp-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-ntp-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-ntp-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-ntp-slave
                namespace: openshift-ntp
              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
    .....

```

5.2. 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オーバーヘッドが発生します。CR は **PolicyGenTemplate** CR の **policyName** フィールドに基づいてポリシーに統合されます。**policyName** に同じ値を持つ同じ **PolicyGenTemplate** の CR は単一のポリシーで管理されます。
- 切断された環境では、すべての Operator を含む単一のインデックスとしてレジストリーを設定することにより、すべての Operator に対して単一のカタログソースを使用します。マネージドクラスターに **CatalogSource** CR を追加するたびに、CPU 使用率が増加します。
- **MachineConfig** CR は、インストール時に適用されるように **SiteConfig** CR に **追加の Manifest** として組み込む必要があります。これにより、クラスターがアプリケーションをデプロイする準備ができるまで全体的な時間がかかる可能性があります。
- **PolicyGenTemplates** は、必要なバージョンを明示的に指定するために channel フィールドを上書きする必要があります。これにより、アップグレード時にソース CR が変更されても、生成されたサブスクリプションが更新されないようになります。

関連情報

- RHACM を使用したクラスターのスケーリングに関する推奨事項は、[パフォーマンスおよびスケーラビリティ](#) 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。

5.3. RAN デプロイメントの POLICYGENTEMPLATE CR

PolicyGenTemplate (PGT) カスタムリソース (CR) を使用して、GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してクラスターに適用される設定をカスタマイズします。PGT CR を使用すると、1つ以上のポリシーを生成して、クラスターのフリートで設定 CR のセットを管理できます。PGT は、管理された CR のセットを識別し、それらをポリシーにバンドルし、それらの CR をラップするポリシーを構築し、ラベルバインディングルールを使用してポリシーをクラスターに関連付けます。

GitOps ZTP コンテナ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。バリエーションは、単一ノード、3ノードのコンパクト、および標準のクラスター設定の相違点に対応するために、グループ **PolicyGenTemplate** CR に提供されます。同様に、シングルノードクラスターとマルチノード (コンパクトまたはスタンダード) クラスターについても、サイト固有の設定バリエーションが提供されています。デプロイメントに関連するグループおよびサイト固有の設定バリエーションを使用します。

表5.1 RAN デプロイメントの PolicyGenTemplate CR

| PolicyGenTemplate CR | 説明 |
|------------------------------------|--|
| example-multinode-site.yaml | マルチノードクラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。 |
| example-sno-site.yaml | 単一ノードの OpenShift クラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。 |

| PolicyGenTemplate CR | 説明 |
|--|---|
| 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リプションします。 |
| group-du-3node-ranGen.yaml | 3 ノードクラスター用の RAN ポリシーのみが含まれています。 |
| group-du-sno-ranGen.yaml | シングルノードクラスター用の RAN ポリシーのみが含まれています。 |
| group-du-standard-ranGen.yaml | 標準的な 3 つのコントロールプレーンクラスターの RAN ポリシーが含まれています。 |
| group-du-3node-validator-ranGen.yaml | PolicyGenTemplate CR は、3 ノードクラスターに必要なさまざまなポリシーを生成するために使用されます。 |
| group-du-standard-validator-ranGen.yaml | 標準クラスターに必要なさまざまなポリシーを生成するために使用される PolicyGenTemplate CR。 |
| group-du-sno-validator-ranGen.yaml | PolicyGenTemplate CR は、単一ノードの OpenShift クラスターに必要なさまざまなポリシーを生成するために使用されます。 |

関連情報

- [GitOps ZTP サイト設定リポジトリの準備](#)

5.4. 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 リポジトリを作成しています。リポジトリ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** フォルダから選択します (**example-sno-site.yaml** または **example-multinode-site.yaml**)。
 - b. サンプルファイルの **bindingRules** フィールドを、**SiteConfig** CR に含まれるサイト固有のラベルと一致するように変更します。サンプルの **SiteConfig** ファイルでは、サイト固有のラベルは **sites: example-sno** です。



注記

PolicyGenTemplate bindingRules フィールドで定義されているラベルが、関連するマネージドクラスターの **SiteConfig** CR で定義されているラベルに対応していることを確認してください。

- c. サンプルファイルの内容を目的の設定に合わせて変更します。
2. オプション: クラスターのフリート全体に適用される一般的な設定 CR の **PolicyGenTemplate** CR を作成します。
 - a. **out/argocd/example/policygentemplates** フォルダから CR の適切な例を選択します (例: **common-ranGen.yaml**)。
 - b. サンプルファイルの内容を目的の設定に合わせて変更します。
3. オプション: フリート内のクラスターの特定のグループに適用されるグループ設定 CR の **PolicyGenTemplate** CR を作成します。

オーバーレイ仕様ファイルの内容が必要な終了状態と一致することを確認します。

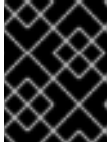
out/source-crs ディレクトリーには、**PolicyGenTemplate** テンプレートに含めることができる **source-crs** の完全な一覧が含まれます。



注記

クラスターの特定の要件に応じて、クラスターの種類ごとに1つ以上のグループポリシーが必要になる場合があります。特に、サンプルのグループポリシーにはそれぞれ単一の **PerformancePolicy.yaml** ファイルがあり、それらのクラスターが同一のハードウェア設定である場合にのみクラスターのセット全体で共有できることを考慮しています。

- a. **out/argocd/example/policygentemplates** フォルダから CR の適切な例を選択します (例: **group-du-sno-ranGen.yaml**)。
 - b. サンプルファイルの内容を目的の設定に合わせて変更します。
4. オプション: GitOps ZTP のインストールとデプロイされたクラスターの設定が完了したときに通知するバリデータ通知ポリシー **PolicyGenTemplate** CR を作成します。詳細は、バリデータ通知ポリシーの作成を参照してください。
5. **out/argocd/example/policygentemplates/ns.yaml** ファイルの例と同様の YAML ファイルで、すべてのポリシーの namespace を定義してください。



重要

Namespace CR を **PolicyGenTemplate** CR と同じファイルに含めないでください。

6. `out/argocd/example/policygentemplates/kustomization.yaml` に示されている例と同様に、**PolicyGenTemplate** CR と **Namespace** CR をジェネレーターセクションの `kustomization.yaml` ファイルに追加します。
7. **PolicyGenTemplate** CR、**Namespace** CR、および関連する `kustomization.yaml` ファイルを Git リポジトリにコミットし、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 ZTP クラスターデプロイメントの完了のシグナリング](#)

5.5. マネージドクラスターポリシーのデプロイメントの進行状況の監視

ArgoCD パイプラインは、Git の **PolicyGenTemplate** CR を使用して RHACM ポリシーを生成し、ハブクラスターに同期します。支援されたサービスが OpenShift Container Platform をマネージドクラスターにインストールした後、管理対象クラスターのポリシー Synchronization の進行状況をモニターできます。

前提条件

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

手順

1. Topology Aware Lifecycle Manager (TALM) は、クラスターにバインドされている設定ポリシーを適用します。
クラスターのインストールが完了し、クラスターが **Ready** になると、`ran.openshift.io/ztp-deploy-wave` アノテーションで定義された順序付きポリシーのリストで、このクラスターに対応する **ClusterGroupUpgrade** CR が TALM により自動的に作成されます。クラスターのポリシーは、**ClusterGroupUpgrade** CR に記載されている順序で適用されます。

以下のコマンドを使用して、設定ポリシー調整のハイレベルの進捗を監視できます。

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

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[-1:]}' | jq
```

出力例

```
{
  "lastTransitionTime": "2022-11-09T07:28:09Z",
  "message": "Remediating non-compliant policies",
```

```

"reason": "InProgress",
"status": "True",
"type": "Progressing"
}

```

2. RHACM ダッシュボードまたはコマンドラインを使用して、詳細なクラスターポリシーのコンプライアンスステータスを監視できます。

a. **oc** を使用してポリシーのコンプライアンスを確認するには、次のコマンドを実行します。

```
$ oc get policies -n $CLUSTER
```

出力例

| NAME | REMEDIATION ACTION | COMPLIANCE STATE | |
|--|--------------------|------------------|-------|
| AGE | | | |
| ztp-common.common-config-policy | inform | Compliant | |
| 3h42m | | | |
| ztp-common.common-subscriptions-policy | inform | NonCompliant | |
| 3h42m | | | |
| ztp-group.group-du-sno-config-policy | inform | NonCompliant | |
| 3h42m | | | |
| ztp-group.group-du-sno-validator-du-policy | inform | NonCompliant | |
| 3h42m | | | |
| ztp-install.example1-common-config-policy-pjz9s | enforce | Compliant | |
| 167m | | | |
| ztp-install.example1-common-subscriptions-policy-zzd9k | enforce | NonCompliant | |
| 164m | | | |
| ztp-site.example1-config-policy | inform | NonCompliant | 3h42m |
| ztp-site.example1-perf-policy | inform | NonCompliant | 3h42m |

b. RHACM Web コンソールからポリシーのステータスを確認するには、次のアクションを実行します。

i. **ガバナンス → ポリシーの検索** をクリックします。

ii. クラスターポリシーをクリックして、ステータスを確認します。

すべてのクラスターポリシーが準拠すると、クラスターの GitOps ZTP のインストールと設定が完了します。**ztp-done** ラベルがクラスターに追加されます。

参照設定では、準拠する最終的なポリシーは、***-du-validator-policy** ポリシーで定義されたものです。このポリシーは、クラスターに準拠する場合、すべてのクラスター設定、Operator のインストール、および Operator 設定が完了します。

5.6. 設定ポリシー CR の生成の検証

ポリシーのカスタムリソース (CR) は、作成元の **PolicyGenTemplate** と同じネームスペースで生成される。以下のコマンドを使用して示すように、**ztp-common**、**ztp-group**、または **ztp-site** ベースのいずれであるかにかかわらず、**PolicyGenTemplate** から生成されたすべてのポリシー CR に同じトラブルシューティングフローが適用されます。

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

予想される policy-wrapped CR のセットが表示されるはずですが。

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。例えば、無効な **sourceFile** → **fileName:** を設定すると、以下のようなエラーが発生します。

```
Status:
Conditions:
  Last Transition Time: 2021-11-26T17:21:39Z
  Message:          rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or directory
Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit status 1: exit
status 1
  Type: ComparisonError
```

3. **Status: Sync:** をチェックします。**Status: Conditions::** でログエラーが発生した場合 **Status: Sync:** に **Unknown** または **Error** と表示されます。

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

4. Red Hat Advanced Cluster Management (RHACM) が **ManagedCluster** オブジェクトにポリシーが適用されることを認識すると、ポリシー CR オブジェクトがクラスターネームスペースに適用されます。ポリシーがクラスターネームスペースにコピーされたかどうかを確認します。

```
$ oc get policy -n $CLUSTER
```

出力例:

| NAME | REMEDIATION ACTION | COMPLIANCE STATE | AGE |
|--|--------------------|------------------|-----|
| ztp-common.common-config-policy | inform | Compliant | 13d |
| ztp-common.common-subscriptions-policy | inform | Compliant | 13d |

| | | | |
|--|--------|-----------|-----|
| ztp-group.group-du-sno-config-policy | inform | Compliant | 13d |
| Ztp-group.group-du-sno-validator-du-policy | inform | Compliant | 13d |
| ztp-site.example-sno-config-policy | inform | Compliant | 13d |

RHACM は、適用可能なすべてのポリシーをクラスターの namespace にコピーします。コピーされたポリシー名の形式は `<policyGenTemplate.Namespace>.<policyGenTemplate.Name>-<policyName>` です。

5. クラスター namespace にコピーされないポリシーの配置ルールを確認します。これらのポリシーの **PlacementRule** の **matchSelector**、**ManagedCluster** オブジェクトのラベルと一致する必要があります。

```
$ oc get placementrule -n $NS
```

6. **PlacementRule** 名は、以下のコマンドを使用して、不足しているポリシー (common、group、または site) に適した名前であることを注意してください。

```
$ oc get placementrule -n $NS <placementRuleName> -o yaml
```

- status-decisions にはクラスター名が含まれている必要があります。
- spec の **matchSelector** の key-value ペアは、マネージドクラスター上のラベルと一致する必要があります。

7. 以下のコマンドを使用して、**ManagedCluster** オブジェクトのラベルを確認します。

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. 以下のコマンドを使用して、準拠しているポリシーを確認します。

```
$ oc get policy -n $CLUSTER
```

Namespace、**OperatorGroup**、および **Subscription** ポリシーが準拠しているが Operator 設定ポリシーが該当しない場合、Operator はマネージドクラスターにインストールされていない可能性があります。このため、スポークに CRD がまだ適用されていないため、Operator 設定ポリシーの適用に失敗します。

5.7. ポリシー調整の再開

たとえば、**ClusterGroupUpgrade** カスタムリソース (CR) がタイムアウトした場合など、予期しないコンプライアンスの問題が発生した場合は、ポリシー調整を再開できます。

手順

1. **ClusterGroupUpgrade** CR は、管理クラスターの状態が **Ready** になった後に Topology Aware Lifecycle Manager によって namespace **ztp-install** に生成されます。

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

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. 予期せぬ問題が発生し、設定されたタイムアウト (デフォルトは 4 時間) 内にポリシーが苦情にならなかった場合、**ClusterGroupUpgrade** CR のステータスは **UpgradeTimedOut** と表示されます。

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Ready")]'}
```

3. **UpgradeTimedOut** 状態の **ClusterGroupUpgrade** CR は、1 時間ごとにポリシー照合を自動的に再開します。ポリシーを変更した場合は、既存の **ClusterGroupUpgrade** CR を削除して再試行をすぐに開始できます。これにより、ポリシーをすぐに調整する新規 **ClusterGroupUpgrade** CR の自動作成がトリガーされます。

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

ClusterGroupUpgrade CR が **UpgradeCompleted** のステータスで完了し、管理対象のクラスターに **ztp-done** ラベルが適用されると、**PolicyGenTemplate** を使用して追加の設定変更を行うことができます。既存の **ClusterGroupUpgrade** CR を削除しても、TALM は新規 CR を生成しません。

この時点で、GitOps ZTP はクラスターとの対話を完了しました。それ以降の対話は更新として扱われ、ポリシーの修復のために新しい **ClusterGroupUpgrade** CR が作成されます。

関連情報

- Topology Aware Lifecycle Manager (TALM) を使用して独自の **ClusterGroupUpgrade** CR を作成する方法は、[ClusterGroupUpgrade CR について](#) を参照してください。

5.8. 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 | COMPLIANCE STATE | AGE | REMEDIATION ACTION |
|-----------|--|------------------|-----|--------------------|
| default | cgu-ztp-group.group-du-sno-config-policy | enforce | 17m | |
| default | ztp-group.group-du-sno-config-policy | inform | 15h | NonCompliant |

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。

5.9. GITOPS ZTP インストール完了の表示

GitOps Zero Touch Provisioning (ZTP) は、クラスターの GitOps ZTP インストールステータスを確認するプロセスを単純化します。GitOps ZTP ステータスは、クラスターのインストール、クラスター設定、GitOps ZTP 完了の3つのフェーズを遷移します。

クラスターインストールフェーズ

クラスターのインストールフェーズは、**ManagedCluster** CR の **ManagedClusterJoined** および **ManagedClusterAvailable** 条件によって示されます。**ManagedCluster** CR にこの条件がない場合や、条件が **False** に設定されている場合、クラスターはインストールフェーズに残ります。インストールに関する追加情報は、**AgentClusterInstall** および **ClusterDeployment** CR から入手できます。詳細は、Troubleshooting GitOps ZTP を参照してください。

クラスター設定フェーズ

クラスター設定フェーズは、クラスターの **ManagedCluster** CR に適用される **ztp-running** ラベルで示されます。

GitOps ZTP 完了

クラスターのインストールと設定は、GitOps ZTP 完了フェーズで実行されます。これは、**ztp-running** ラベルを削除し、**ManagedCluster** CR に **ztp-done** ラベルを追加することで表示されます。**ztp-done** ラベルは、設定が適用され、ベースライン DU 設定が完了したことを示しています。ZTP 完了状態への遷移は、Red Hat Advanced Cluster Management (RHACM) バリデーターのインフォームドポリシーの準拠状態が条件となります。このポリシーは、完了したインストールの既存の基準をキャプチャし、マネージドクラスターの GitOps ZTP プロビジョニングが完了したときのみ、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 デモンセットが存在する。

第6章 ZTP を使用した単一ノード OPENSIFT クラスターの手動インストール

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



注記

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



重要

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

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

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

前提条件

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

手順

1. 次のコマンドを実行して、出力フォルダーを作成します。

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

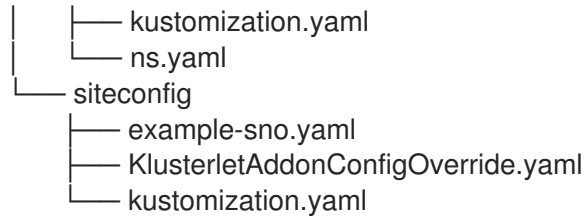
2. **ztp-site-generate** コンテナイメージから **argocd** ディレクトリーをエクスポートします。

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.15 extract /home/ztp --tar | tar x -C ./out
```

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

出力例

```
out
├── argocd
│   └── example
│       ├── policygentemplates
│       │   ├── common-ranGen.yaml
│       │   ├── example-sno-site.yaml
│       │   ├── group-du-sno-ranGen.yaml
│       │   └── group-du-sno-validator-ranGen.yaml
```



3. サイトインストール CR の出力フォルダーを作成します。

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

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

```

# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created under
# same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
    - clusterName: "example-sno"
      networkType: "OVNKubernetes"
      # installConfigOverrides is a generic way of passing install-config
      # parameters through the siteConfig. The 'capabilities' field configures
      # the composable openshift feature. In this 'capabilities' setting, we
      # remove all but the marketplace component from the optional set of
      # components.
      # Notes:
      # - NodeTuning is needed for 4.13 and later, not for 4.12 and earlier
      installConfigOverrides: "{\"capabilities\":{\"baselineCapabilitySet\": \"None\",
      \"additionalEnabledCapabilities\": [ \"marketplace\", \"NodeTuning\" ] }}"
      # 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: "4.14"
        # 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

```

```

# ../policytemplates/group-du-sno-ranGen.yaml will apply to all clusters with 'group-
du-sno: ""
group-du-sno: ""
# ../policytemplates/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:
    wwn: "0x11111000000asd123"
    # example of diskPartition below is used for image registry (check ImageRegistry.md
for more details), but it's not limited to this use case
    # diskPartition:
    # - device: /dev/disk/by-id/wwn-0x11111000000asd123 # match rootDeviceHints
    # partitions:
    # - mount_point: /var/imageregistry
    # size: 102500
    # start: 344844

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** コンテナからマニフェストを取得しません。

- 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:v4.15 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:v4.15 generator install -E site-1-sno.yaml /output
```

出力例

```

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

```

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

- a. Day 2 CR の出力フォルダーを作成します。

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

- b. Day 2 設定 CR を生成してエクスポートします。

```
$ podman run -it --rm -v `pwd`/out/argocd/example/policygentemplates:/resources:Z -v `pwd`/ref:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.15 generator config -N . /output
```

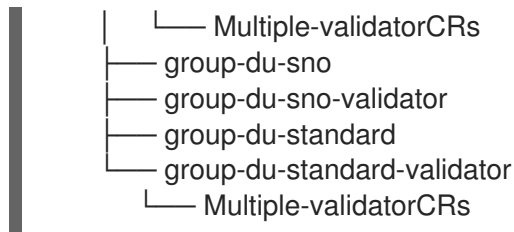
このコマンドは、単一ノード OpenShift、3 ノードクラスター、および標準クラスター用のサンプルグループおよびサイト固有の **PolicyGenTemplate** CR を **./ref** フォルダーに生成します。

出力例

```

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

```

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,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= ❶
        node-role.kubernetes.io/master=
        node-role.kubernetes.io/worker=
        node.openshift.io/os_id=rhcos
```

- ❶ 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](#)

6.2. 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 ❶
data: ❷
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno ❸
data:
  .dockerconfigjson: <pull_secret> ❹
type: kubernetes.io/dockerconfigjson

```

- ❶ 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
- ❷ **password** と **username** の Base64 エンコード値
- ❸ 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
- ❹ 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。

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

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



注記

OpenShift Container Platform 4.15 では、カーネル引数の追加のみを行うことができません。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
```

6.4. 単一のマネージドクラスターのインストール

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

前提条件

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

手順

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

```
apiVersion: hive.openshift.io/v1
kind: ClusterImageSet
metadata:
  name: openshift-4.15.0 ①
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.15.0-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-4.15.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** コンテナから抽出し、要件を満たすようにカスタマイズした、生成された day-0 CR を適用します。

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

関連情報

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

6.5. 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 **AgentsConnected** 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
```

6.6. 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。

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

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



注記

すべてのマネージドクラスターには独自の namespace があり、**ManagedCluster** と **ClusterImageSet** を除くすべてのインストール CR はその namespace の下にありません。**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6.1 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 |
|----------------------------|---|--|
| BareMetal Host | ターゲットのベアメタルホストの Baseboard Management Controller (BMC) の接続情報が含まれています。 | Redfish プロトコルを使用して、BMC へのアクセスを提供し、ターゲットサーバーで検出イメージをロードおよび開始します。 |
| InfraEnv | ターゲットのベアメタルホストに OpenShift Container Platform をインストールするための情報が含まれています。 | ClusterDeployment で使用され、マネージドクラスターの Discovery ISO を生成します。 |
| AgentClusterInstall | ネットワークやコントロールプレーンノードの数など、マネージドクラスター設定の詳細を指定します。インストールが完了すると、クラスター kubeconfig と認証情報が表示されます。 | マネージドクラスターの設定情報を指定し、クラスターのインストール時にステータスを指定します。 |

| CR | 説明 | 使用法 |
|------------------------------|---|--|
| 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。 | <ul style="list-style-type: none"> ● 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、ターゲットベアメタルホストにインストールされている OpenShift Container Platform イメージの認証情報が含まれます。 |
| ClusterImageSet | リポジトリ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7章 VDU アプリケーションのワークロードに推奨される単一ノードの OPENSIFT クラスタ設定

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にデプロイするために必要な単一ノードの OpenShift 設定を理解してください。設定には、高性能ワークロードのためのクラスタの最適化、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

- 単一クラスタを手動でデプロイするには、[GitOps ZTP を使用した単一ノード OpenShift クラスタ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のフリートをデプロイするには、[GitOps ZTP を使用した遠端サイトのデプロイ](#) を参照してください。

7.1. OPENSIFT 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。

7.2. 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7.1 最小リソース要件

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



注記

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

- $(\text{コアあたりのスレッド数} \times \text{コア}) \times \text{ソケット} = \text{vCPU}$



重要

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

7.3. 低遅延と高パフォーマンスのためのホストファームウェアの設定

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

手順

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



重要

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

表7.2 Intel Xeon Skylake または Cascade Lake サーバーのファームウェア設定例

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

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



注記

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

7.4. マネージドクラスタ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。

- ハブクラスタ内の GitOps ZTP コンテナ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**
- ハブクラスタは、マネージドクラスタの 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**

7.5. GITOPS ZTP を使用した単一ノードの OPENSIFT 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、クラスタのインストールに使用する **SiteConfig** カスタムリソース (CR) の **cpuPartitioningMode** フィールドを設定し、ホスト上で **isolated** と **reserved** CPU を設定する **PerformanceProfile** CR を適

用します。

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

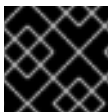


注記

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

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

ワークロードパーティショニング設定は、OpenShift Container Platform インフラストラクチャー Pod を **reserved** CPU セットに固定します。systemd、CRI-O、kubelet などのプラットフォームサービスは、**reserved** CPU セット上で実行されます。**isolated** CPU セットは、コンテナ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争することなく、ワークロードが指定された CPU に確実にアクセスできるようになります。分離されていないすべての CPU を予約する必要があります。



重要

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

関連情報

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

7.6. 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。

7.6.1. ワークロードの分割

DU ワークロードを実行する単一ノードの OpenShift クラスタには、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。



OPENSIFT 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、クラスタのプロビジョニングに使用する **/extra-manifest** フォルダ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 ❶
```

- ❶ クラスタ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。

- マネージドクラスタ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
```

- 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
```



```

CgppZiBbWyAkRU5WRkIMRSBdXTsgdGhIbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUI
R9CiAgZWNoYAiJHtWQVJOQU1Fft0ke0VYRUNTVEFSVH0iID4gJEVOVkJZTEUKZWxzZQogIGVja
G8gJEVYRUNTVEFSVApmaQo=
  mode: 493
  path: /usr/local/bin/extractExecStart
- contents:
  source: data:text/plain;charset=utf-
8;base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbWV
zcGFjZS9tbnQgliRAIgo=
  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="OPENSIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
  Environment="OPENSIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
  name: 30-kubelet-interval-tuning.conf
name: kubelet.service

```

7.6.3. SCTP

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

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

```

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

```

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

```

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

```


cmUgYXJIIHNvbWUgcGVyaW9kcyB3aGVyZSB0aGUKlyBjb3VudCBkb2Vzbid0IGluY3JlYXNlIGJ1dCB3
ZSBrbm93IHdlIGNhbD0IGJlIGF0IHNOZWFkeS1zdGF0ZSB5ZXQuCINURUFUFEWV9TVEFURV9NSU5J
TVVNPSR7U1RFQURZX1NUQVRFXX01JTKINUVU06LTB9CgoylyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjly
yMjly
dV9tYW5hZ2VyX3NOYXRICKZVTExfQ1BVX1NUQVRFPS9zeXMvZnMvY2dyb3VwL2NwdXNldC9jcHVz
ZXQuY3B1cwpLVUJFTEVUX0NPTkY9L2V0Yy9rdWJlcm5ldGVzL2t1YmVsZXQuY29uZgp1bnJlc3Rya
WN0ZWRDcHVzZXQoKSB7CiAgbG9jYwWgY3B1cwogIGlmIFtbcIC1IICRLVUJFTEVUX0NQQV9TVEF
URSBdXTsgdGhIbgogICAgY3B1cz0kKGpxIC1yICcuZGVmYXVsdENwdVNldCcgPCRLVUJFTEVUX0N
QVV9TVEFURSkKICAgIglmIFtbcIC1ulClke2NwdXN9liAmJiAtZSAke0tVQkVMRVRfQ09ORn0gXV07IH
RoZW4KICAgIcAgcmVzZXJ2ZWRfY3B1cz0kKGpxIC1yICcuZGVmYXVsdENwdVNldCcgPCRLVUJFTEVUX0N
V0Yy9rdWJlcm5ldGVzL2t1YmVsZXQuY29uZikKICAgICAgWYgW1sgLW4gliR7cmVzZXJ2ZWRfY3B1c
3oiIF1dOyB0aGVuCiAgICAgICAglyBVc2UgdGFza3NldCB0byBtZXJnZSB0aGUgdHdVIGNwdXNldHMK
ICAgICAgICBjcHVzPSQodGFza3NldCAAtYyAiiHtyZXNlcnZlZf9jcHVzFswke2NwdXN9liBncmVwIC1pIE
NwdXNfYWxsb3dlZF9saXNOIC9wcm9jL3NlbgYvc3RhdHVzIHwgYXdrICd7cHJpbnQgJDJ9JykKICAgIC
AgZmkKICAgICAgZpCiAgZmkKICBpZiBbWyAteiAkY3B1cyBdXTsgdGhIbgogICAglyBmYWxslGJhY2sgdG
8gdXNpbmcmYwXslGNwdXMgaWYgdGhlIGt1YmVsZXQgc3RhdGUgaXMgY3B1c2V0KICAgICAgZm90IGNvbMzZ3VyZWQ
geWV0CiAgICBbWyAtZSAkRiVMtF9DUFVfU1RBVEUgXV0gfHwgcmV0dXJlIDEKICAgICAgNwdXM9JC
g8JEZVTExfQ1BVX1NUQVRFKQogIGZpCiAgZWNobyAkY3B1cwp9CgpyZXN0cmliGvKQ3B1c2V0K
CkgewogIGZvciBhcmcmgaW4gJCg8L3Byb2MvY21kbGluZSk7IGRvCiAgICBpZiBbWyAkYXJnID1tIF5zeX
NOZW1kLmNwdV9hZmZpbml0eT0gXV07IHRoZW4KICAgICAgZWNobyAke2FyZyMqPX0KICAgICAgc
mV0dXJlDAKICAgICAgZpCiAgZG9uZQogIHJldHVybiAxXn0KCnJlc2V0QWZmaW5pdHkoKSB7CiAgbG
9jYwWgY3B1c2V0PSIkMSIKICBsb2NhbCBmYwYsY291bnQ9MAogIGxvY2FsIHN1Y2Nlc3Njb3VudD0w
CiAgbG9nZ2VyICJSZWNvdmVyeTogU2V0dGluZyBDUFUgYWZmaW5pdHkgZm9yIGNyaXRpY2FsIH
Byb2Nlc3NlcyBcliRDUKIUSUNBTF9QUk9DRVNTRVNcllB0byAkY3B1c2V0lgogIGZvciBwcm9jIGluCRD
UkiUSUNBTF9QUk9DRVNTRVM7IGRvCiAgICBsb2NhbCBwaWRzPSIkKHBncmVwICRwcm9jKSIKIC
AgICAgICAglyBwaWQgaW4gJHBpZHM7IGRvCiAgICAgICAglyBwY2FsIHhrhc2tzXZRPdXRwdXQKICAgICAgdG
Fza3NldE91dHB1dD0iJCh0YXNrc2V0IC1hcGmglirjchVzZXQilCRwaWQgMj4mMSkiCiAgICAgIClmIFt
bICQ/IC1uZSAwF1dOyB0aGVuCiAgICAgICAgZWNobyAiiRVJST1I6ICR0YXNrc2V0T3V0cHV0lgogIC
AgICAgICAgCgoZmFpbGNvdW50KyspKQogICAgICBibHNlCiAgICAgICAgICAgKChzdWNjZXNzY291bnQrKykp
CiAgICAgICAgICAgZpCiAgICBkb25lCiAgZG9uZQoKICBsb2dnZXIglJlY292ZXJ5OisBSZS1hZmZpbmVklCRzd
WNjZXNzY291bnQgcGkkyBzdWNjZXNzZnVsbHkiCiAgWYgW1sgJGZhaWxjb3VudCAAtZ3QgMCBdXT
sgdGhIbgogICAgbG9nZ2VyICJSZWNvdmVyeTogRmFpbGVkIHRRvIHJLWFmZmluZSAkZmFpbGNvdW
50IHByb2Nlc3NlcyIKICAgIHJldHVybiAxCiAgZmkKfQoKc2V0VW5yZXN0cmliGvKkCkgewogIGxvZ2dlci
AiUmVjb3Zlcnk6IFNldHRpbmcmY3JpdGlyYWwgc3lzdGVtIHByb2Nlc3NlcyB0byBoYXZlIHVucmVzdHJpYy
RIZCBDUFUgYWNjZXNzlgogIHJlc2V0QWZmaW5pdHkggliQodW5yZXN0cmliGvKQ3B1c2V0KSIKfQo
Kc2V0UmVzdHJpY3RIZCgplHsKICBsb2dnZXIglJlY292ZXJ5OisBSZSZNldHRpbmcmY3JpdGlyYWwgc3lzd
GVtIHByb2Nlc3NlcyBiYWNrIHRRvIG5vcm1hbGx5IHJlc3RyaWN0ZWQgYWNjZXNzlgogIHJlc2V0QWZm
aW5pdHkggliQocmVzdHJpY3RIZENwdXNldCkiCn0KCmN1cnJlbnRBZmZpbml0eSgplHsKICBsb2NhbC
BwaWQ9liQxlqogIHRhc2tzXZQgLBjICRwaWQgfCBhd2sgLUYnOiaAnIcd7cHJpbnQgJDJ9Jwp9Cgp3a
XRoaW4oKSB7CiAgbG9jYwWgY3B1cwogIGlmIFtbcIC1IICRLVUJFTEVUX0NQQV9TVEFURSBdXTsgdGhIbgog
ICAgG9nZ2VyICJSZWNvdmVyeTogU2V0dGluZyBDUFUgYWZmaW5pdHkgZm9yIGNyaXRpY2FsIH
GRIbHRhPTAgcGN0YW5nZQogIGRIbHRhPSQoKCBjdXJyZW50IC0gbGFzdCApKQogIGlmIFtbcICRjld
XJyZW50IC1cSAkbGFzdCBdXTsgdGhIbgogICAgcGN0YW5nZT0wCiAgZWNpZiBbWyAkY3B1c2V0KICAgICAgdG
EgMCBdXTsgdGhIbgogICAgcGN0YW5nZT0xMDAwMDAwCiAgZWNpZiBbWyAkY3B1c2V0KICAgICAgdG
gKCAkZGVsdGEgKiAxMDApIC8gbGFzdCApKQogIGZpCiAgZWNobyAtbiAibGFzdDokbGFzdCBjdXJy
ZW50OiRjdXJyZW50IGRIbHRhOiRkZWx0YSBwY2hhbmdlOiR7cGN0YW5nZX0iOiAiCiAgbG9jYwWgY
WJzb2x1dGUgbGltaxQKICBjYXNlICR0aHJlc2hvbGQgaW4KICAgIColKQogICAgICBhYnNvbHV0ZT0k
e3BjaGFuZ2Ujly19ICMgYwWJzb2x1dGUgdmFsdWUKICAgICAgbGltaxQ9JHt0aHJlc2hvbGQlJSV9CiAgI
cAgICds7CiAgICAgKQogICAgICBhYnNvbHV0ZT0ke2RlbHRhlyMtfSAjIGFic29sdXRlIHZhbHVlCiAgICAg
ICAgICAglyBwWl0PSR0aHJlc2hvbGQKICAgICAgOzsKICBlc2FjCiAgWYgW1sgJGFic29sdXRlIC1sZSAkbGltax
QgXV07IHRoZW4KICAgICAglyBjaG8gIndpdGhpbiAoKy8tKSR0aHJlc2hvbGQlCiAgICAgICByZXR1cm4gMAogI
GVsc2UKICAgICAglyBjaG8gIm91dHNpZGUgKCSvLSkKdGhyZXNob2xklqogICAgcmV0dXJlIDEKICAgICAgmaQ
p9CgpdzGVhZlhdGF0ZSgplHsKICBsb2NhbCBsYXN0PSQxIGN1cnJlbnQ9JDlKICBpZiBbWyAkY3B1c2V0K
dCAAtbHQgJFNURUFUFEWV9TVEFURV9NSU5JTVVNIF1dOyB0aGVuCiAgICAgICAglyBhY2hvcICsYXN0OiRsYXN
0IGN1cnJlbnQ6JGN1cnJlbnQgV2FpdGluZyB0byByZWVjaCAKU1RFQURZX1NUQVRFXX01JTKINUVU0g
YmVmb3JlIGNoZWNraW5nIGZvciBzdGVhZkKc3RhdGUiCiAgICAgICByZXR1cm4gMQogIGZpCiAgd2l0aGI
ulCRsYXN0ICRjldXJyZW50ICRTVEVBRfifU1RBVEVfVEhSRVNIT0xECn0KCndhaXRGb3JlSzwfkeSg


```
# 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: accelerated-container-startup.service
- contents: |
  [Unit]
  Description=Unlocks more CPUs for critical system processes during container shutdown
  DefaultDependencies=no

  [Service]
  Type=simple
  ExecStart=/usr/local/bin/accelerated-container-startup.sh

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

# Steady-state threshold
# 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=-1

# Steady-state window = 60s
# 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=60

[Install]
WantedBy=shutdown.target reboot.target halt.target
enabled: true
name: accelerated-container-shutdown.service
```

7.6.5. rcu_normal の設定

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

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 08-set-rcu-normal-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,lyEvYmluL2Jhc2gKIwojIERpc2FibGUgcmN1X2V4cGVkaXRIZCBhZnRlciBub2RlIGhhcyBmaW
5pc2hlZCBib290aW5nCiMKlyBUaGUgZGVmYXVsdHMgYmVsb3cgY2FuIGJlIG92ZXJyaWRkZW4gdml
hIGVudmlyb25tZW50IHZhcmlhYmxlcwojCgojIERIZmF1bHQgd2FpdCB0aW1lIGlzlDYwMHMgPSAxMG
06Ck1BWEINVU1fV0FJVf9USU1FPSR7TUFYSU1VTV9XQUiUX1RJTUU6LTYwMH0KCIMgRGVmY
XVsdCBzdGVhZHk3RhdGUgdGhyZXNob2xklD0gMiUKlyBBbGxvd2VklHZhbHVlcz0klyAgNCAgLSBh
YnNvbHV0ZSBwb2QgY291bnQgKCsLskKlyAgNCUgLSBwZXJjZW50IGNoYW5nZSAoKy8tKQojICAt
MSAtIGRpc2FibGUgdGhIHhN0ZWFkeS1zdGF0ZSBjaGVjapwTVEVBRFifU1RBVEVfVEhSRVNIT0xEP
SR7U1RFQRZX1NUQVRFX1RIUkVTSE9MRDotMiV9CgojIERIZmF1bHQgd3RIYWRR5LXN0YXRlIHd
pbmRvdyA9IDYwcmwojElmIHRoZSBYdW5uaW5nIHVZCBjb3VudCBzdGF5cyB3aXRoaW4gdGhllGdpd
mVulHRocmVzaG9sZCBmb3IgdGhpcyB0aW1lCiMgcGVyaW9kLCByZXRx1cm4gQ1BVIHV0aWxpemF0
aW9uHRVlG5vcm1hbCBiZWZvcUgdGhllG1heGltdW0gd2FpdCB0aW1lIGhhcwojIGV4cGlyZXMKU1
RFQRZX1NUQVRFX1dJTkRPVz0ke1NURUFEWV9TVEFURV9XSU5ET1c6LTYwfQoKlyBEZWZhd
Wx0IHhN0ZWFkeS1zdGF0ZSBhbGxvd3MgYW55IHVZCBjb3VudCB0byBiZSAic3RIYWRR5LXN0YXRlIgo
jIEluY3JIYXNpbmcmgdGhpcyB3aWxsIHhNraXAgYW55IHhN0ZWFkeS1zdGF0ZSBjaGVja3MgdW50aWwgd
GhllGNvdW50IHJpc2VzIGFib3ZlCiMgdGhpcyBudW1iZlZlG8gYXZvaWQgZmFsc2UgcG9zaXRpdmV
zIGlmIHRoZlJlIGFyZSBzb21lIHBlcmVzZHMgd2hlcUgdGhllCiMgY291bnQgZG9lc24ndCBpbmNyZWF
zZSBidXQgd2Uga25vdyB3ZSBjYW4ndCBiZSBhdCBzdGVhZHk3RhdGUgeWV0LgpTVEVBRFifU1RB
VEVfTUOSU1VTT0ke1NURUFEWV9TVEFURV9NSU5JTUVVNOi0wfQoKlyMjlyMjlyMjlyMjlyMjlyMjly
yMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjly
MjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjlyMjly
udD0kMiB0aHJlc2hvbGQ9JDMKICBsb2NhbCBkZWx0YT0wIHVjaGFuZ2UKICBkZWx0YT0kKCggY3Vy
cmVudCAtIGxhc3QgKSskKICBpZiBbWyAkY3VycmVudCAtZXEgJGxhc3QgXV07IHRoZW4KICAgIHVjaG
FuZ2U9MAogIGVsaWYgW1sgJGxhc3QgLWVxIDAgXV07IHRoZW4KICAgIHVjaGFuZ2U9MTAwMDA
wMAogIGVsc2UKICAgIHVjaGFuZ2U9JCGolCgglRkZWx0YSlgKiAxMDApIC8gbGFzdCApKQogIGZpCi
AgZWNobyAtbiAibGFzdDokbGFzdCBjdxJyZW50OiRjdXJyZW50IGRlbnRlRHRhOiRkZWx0YSBwY2hhbmdl
OiR7cGNoYW5nZX0lOIAiCiAgbG9jYWwgYWwJzb2x1dGUgUGltaXQKICBjYXNlICR0aHJlc2hvbGQgaW4
KICAgIColKQogICAgICBhYnNvbHV0ZT0ke3BjaGFuZ2Ujly19ICMgYwJzb2x1dGUgdMfSDWUKICAgI
CAgbGltaXQ9JHt0aHJlc2hvbGQlJSV9CiAgICAgICAgICs7CiAgICAgICAgICAgICAgICAgICAgICAgICAg
RHlyMtfSAjIGFic29sdXRlIHhZhbHVlCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
gaWYgW1sgJGFic29sdXRlIC1sZSAkbGltXQgXV07IHRoZW4KICAgIGVjaG8gIndpdpGhpbiAoKy8tKSR
0aHJlc2hvbGQiCiAgICBjYXRx1cm4gMAogIGVsc2UKICAgIGVjaG8gIm91dHNpZGUgKCsLskkdGhyZ
XNob2xklgogICAgcmV0dXJlDEKICBmaQp9CgpzdGVhZHlzdGF0ZSgplHsKICBsb2NhbCBsYXN0PS
QxlGN1cnJlbnQ9JDIKICBpZiBbWyAkY3VycmVudCAtbiHQgJFNURUFEWV9TVEFURV9NSU5JTUVVNiF1dO
yB0aGVuCiAgICBjY2hvbGQsYXN0OIRsYXN0IGN1cnJlbnQ6JGN1cnJlbnQgV2FpdGluZyB0byByZWJja
CAKU1RFQRZX1NUQVRFX01JTkINVU0gYmVmb3JlIGNoZWNRaW5nIGZvciBzdGVhZHk3RhdGUi
CiAgICBjYXRx1cm4gMQogIGZpCiAgd2l0aGlucmVudCApZlRjdXJyZW50IiAifNURUFEWV9TVEFU
RV9USFJFU0hPTEQiCn0KCndhaXRGb3JSZWFkeSgplHsKICBsb2dnZlIjly292ZXJ5OXBXYWI0aW
```

```

5nICR7TUFYSU1VTV9XQUIUX1RJTUV9cyBmb3lGdGhllGluaXRpYWxpemF0aW9uIHRvIGNvbXBsZX
RllgogIGxvY2FsIHQ9MCBzPTEwCiAgbG9jYWwgbGFzdENjb3VudD0wIGNjb3VudD0wIHNOZWfkeVN
0YXRIVGltZT0wCiAgd2hpbGUgW1sgJHQgLVw0ICRNQVhJTVVNX1dBVSrVfEINRSBdXTsgZG8KIC
AgIHNSZWVwICRzCiAgICAoKHQgKz0gcykpCiAgICAjIERldGVjdCBzdGVhZHhtc3RhdGUgcG9kIGNvd
W50CiAgICBjY291bnQ9JChjcmldGwgcHMgMj4vZGV2L251bGwgfCB3YyAtbCkKICAgIGlmIFtblCRjY2
91bnQgLWd0IDAQXV0gJiYgc3RIYWR5c3RhdGUgUgliRsYXN0Q2NvdW50IiAiJGNjb3VudCI7IHRoZW4KI
CAGlCAgKChzdGVhZHITdGF0ZVRpbWUgKz0gcykpCiAgICAglVjaG8gIIN0ZWfkeS1zdGF0ZSBmb3
lgJHtzdGVhZHITdGF0ZVRpbWV9cy8ke1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgaWYgW
1sgJHN0ZWfkeVN0YXRIVGltZSAiZ2UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICA
glCAglCBsb2dnZXIglJlJlY292ZXJ5OibTbGVhZHhtc3RhdGUgKCsLSAklU1RFQURZX1NUQVRFX1RI
UkVTSE9MRCkgZm9yICR7U1RFQURZX1NUQVRFX1dJTRPV31zOibEeb25llgogICAgICAglHJldHVy
biAwCiAgICAglGZpCiAgICBlbHNiCiAgICAglGlmIFtblCRzdGVhZHITdGF0ZVRpbWUgLVw0IDAQXV07
IHRoZW4KICAgICAglCBiY2hvlCJSZXNldHRpbmccg3RIYWR5LXNOYXRlIHJpbnVlY292ZXJ5OibEeb25llgogICAgICAglH
0ZWfkeVN0YXRIVGltZT0wCiAgICAglGZpCiAgICBlbGogICAgbGFzdENjb3VudD0kY2NvdW50CiAgZ
G9uZQogIGxvZ2dldiAiUmVjb3Zlcnk6IFJlY292ZXJ5IENvbXBsZXRIIFRpbWVvdXQiCn0KCnNldFJjdU5vc
m1hbCgplHsKICBIY2hvlCJTZR0aW5nIHJjdV9ub3JtYWwgdG8gMSIKICBIY2hvlDEgPiAvc3lzL2tldm5lt
C9yY3Vfbm9ybWFSn0Kcm1haW4oKSB7CiAgd2FpdEzvcjJlYWR5CiAgZWNobyAiV2FpdGluZyBmb3I
gc3RIYWR5IHNOYXRlIHJpbnVlY292ZXJ5OibEeb25llgogICAgICAglHJldHVybiB2s6ICQoYXdrIcd7chJpbnQgaW50KCQxLzMDMDApImgiLCBpbnQoKCQ
xJTM2MDApLzYwKSJtliwgaW50KCQxJTYwKSJzIn0nIC9wcm9jL3VwdGltZSkiCiAgc2V0UmN1Tm9yb
WFsCn0KCmlmIFtblCIke0JBU0hfU09VUkNFWzBdfSlgPSAilHswfSlgXV07IHRoZW4KICBtYWIulClke0
B9lgogIGV4aXQgJD8KZmkK

```

```
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
```

7.6.6. kdump による自動カーネルクラッシュダンプ

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

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 05-kdump-config-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
  storage:
    files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,IyEvdXNyL2Jpbi9lbnYgYmFzaAoKlyBUaGlzIHJjcmlwdCBYZW1vdmVzIHRoZSBpY2UgbW9kdWxllGZyb20ga2R1bXAgaG8gcHJldmVudCBvZHVtcCBmYWlscDZlcyBvbiBjZXJ0YWluIHNIcnZlcnMuCiMgVGhpcyBpcyBhIHRlbXBvcmlsZSB3b3JrYXJvdW5kIGZvcjBSSEVMUExBTi0xMzgyMzYgYW5kIGNhbiBzSBYvdmVklHdoZW4gdGhhdBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9li91c3lvYmluL3NIZCikR1JFUD0iL3Vzci9iaW4vZ3JlcCIkCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcgcHVycG9zZXMKMS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbWZpZy9rZHVtcH0iCiJFTU9WRV9JQ0VfU1RSPSJtb2R1bGVfYmxhY2tsaXN0PWljZSIkCiMgZXhpdCBpZiBmaWxllIGRvZXNuJ3QgZXhpc3QKWyAhIC1mICR7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxllGFscmVhZHVycmVhZXBkYXRIZAoKe0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKlyBUYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcbGlrZSB0aGlzOgojIetEVU1QX0NPTU1BTkRMSU5FX0FQUEVORD0iaXJxcG9sbCBucj9jcHVzPTEgLi4uIGhlc3RfZGZlYWJsZSIkIyBvc2Ugc2VklHRvIG1hdGNolGV2ZXJ5dGhpbmcbYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlfJFTU9WRV9JQ0VfU1RSIHRvIGl0CiR7U0VEfSAtaSAncy9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPSJbXiJdKi8m
```

```
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
mode: 448
path: /usr/local/bin/kdump-remove-ice-module.sh
```

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

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

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-kdump-config-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump-remove-ice-module.service
          contents: |
            [Unit]
            Description=Remove ice module when doing kdump
            Before=kdump.service
            [Service]
            Type=oneshot
            RemainAfterExit=true
            ExecStart=/usr/local/bin/kdump-remove-ice-module.sh
            [Install]
            WantedBy=multi-user.target
  storage:
    files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,IyEvdXNyL2Jpbi9IbnYgYmFzaAoKlyBUaGlzIHJcmldCBYZW1vdmVzIHRoZSBpY2UgbW9k
```



```
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBrc2V0IC14CgpTRUQ9li9i91c3lv
MgVGhpcyBpcyBhIHRlbnRlc3B3b3JrYXJvdW5kIGZvcjBSSEVMUEExBTi0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVklHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9li9i91c3lv
YmluL3NIZCikR1JFUD0iL3Vzci9iaW4vZ3JlcCIkCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcmcGcHVycG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iClJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIkCiMgZXhpdCBpZiBmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mIC
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxlIGFscmVhZkZkYXRIZAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKlyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcmcGblrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucl9jcHVzPTEgLi4ulGhlc3RfZGizYWJsZSIklyBVc2Ugc2VkIHRvIG1hdG
NoIGV2ZXJ5dGhpbmcmcYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBmQgdGhlfJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7U0VEfSAAtaSAncy9eS0RVTVBfQ09NTUFORExJTkvfQVBQRU5EPSJbXiJdKi8m
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
mode: 448
path: /usr/local/bin/kdump-remove-ice-module.sh
```

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

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

7.6.7. 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 クラスタでこの問題が発生しないようにするには、クラスタをインストールする際に CRI-O 削除キャッシュ機能を無効にします。

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-crio-disable-wipe-master
spec:
```

```

config:
  ignition:
    version: 3.2.0
  storage:
    files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAilgo=
          mode: 420
          path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml

```

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

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-crio-disable-wipe-worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAilgo=
            mode: 420
            path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml

```

7.6.8. crun をデフォルトのコンテナランタイムに設定

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 コンテナランタイムとして **crun** を設定します。crun コンテナ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

```

7.7. 推奨されるインストール後のクラスタ設定

クラスタ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、クラスタのインストールに使用する **SiteConfig** CR の **spec.clusters.nodes.bootMode** フィールドを更新することで、単一ノード OpenShift クラスタの 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のデプロイ](#) を参照してください。

7.7.1. 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があります。

推奨される Storage Operator namespace と Operator グループ設定 (StorageNS.yaml, StorageOperatorGroup.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-ntp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ntp-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

```

7.7.2. Operator のサブスクリプション

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

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

Operator サブスクリプションごとに、Operator の取得先であるチャンネルを指定します。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、チャン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
```

7.7.3. クラスターのロギングとログ転送

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

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

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

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

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

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

7.7.4. パフォーマンスプロファイル

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



注記

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

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

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

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
  # matches this name: include=openshift-node-performance-{PerformanceProfile.metadata.name}
  # Also in file 'validatorCRs/informDuValidator.yaml':
  # name: 50-performance-{PerformanceProfile.metadata.name}
  name: openshift-node-performance-profile
  annotations:
    ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "vfio_pci.enable_sriov=1"
    - "vfio_pci.disable_idle_d3=1"
    - "module_blacklist=irdma"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
  pages:
    - size: $size
      count: $count
      node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  nodeSelector:
    node-role.kubernetes.io/$mcp: ""
  numa:
    topologyPolicy: "restricted"
    # To use the standard (non-realtime) kernel, set enabled to false
  realTimeKernel:
    enabled: true
  workloadHints:
    # WorkloadHints defines the set of upper level flags for different type of workloads.
    # See https://github.com/openshift/cluster-node-tuning-operator/blob/master/docs/performanceprofile/performance\_profile.md#workloadhints
    # for detailed descriptions of each item.
    # The configuration below is set for a low latency, performance mode.
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false

```

表7.3 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 |
|-----------------------------|----|
|-----------------------------|----|

| PerformanceProfile CR フィールド | 説明 |
|----------------------------------|---|
| metadata.name | <p>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。</p> <ul style="list-style-type: none"> ● TunedPerformancePatch.yaml の include=openshift-node-performance- \${PerformanceProfile.metadata.name} ● validatorCRs/informDuValidator.yaml の name: 50-performance- \${PerformanceProfile.metadata.name} |
| spec.additionalKernelArgs | <p>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。</p> |
| spec.cpu.isolated | <p>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。</p> <div data-bbox="817 1070 922 1323" style="display: inline-block; vertical-align: top;">  </div> <p style="margin-left: 20px;">重要</p> <p style="margin-left: 20px;">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。</p> |
| spec.cpu.reserved | <p>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。</p> |
| spec.hugepages.pages | <ul style="list-style-type: none"> ● huge page の数 (count) を設定します。 ● huge page のサイズ (size) を設定します。 ● node を hugepage が割り当てられた NUMA ノード (node) に設定します。 |
| spec.realTimeKernel | <p>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。</p> |

| PerformanceProfile CR フィールド | 説明 |
|-----------------------------|---|
| 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。 |

7.7.5. クラスター時間同期の設定

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

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

```

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

```

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

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-sync-time-once-worker
spec:
  config:
    ignition:

```

```

version: 3.2.0
systemd:
  units:
    - contents: |
        [Unit]
        Description=Sync time once
        After=network.service
        [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

```

7.7.6. PTP

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

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

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: slave
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "slave"
      # The interface name is hardware-specific
      interface: $interface
      ptp4IOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4IConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 1
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37

```

```
clockClass 255
clockAccuracy 0xFE
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
```

```

first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0xA0
recommend:
- profile: "slave"
  priority: 4
  match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"

```

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

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "boundary"
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO

```

```
ptpSchedulingPriority: 10
ptpSettings:
  logReduce: "true"
ptp4lConf: |
# The interface name is hardware-specific
[$iface_slave]
masterOnly 0
[$iface_master_1]
masterOnly 1
[$iface_master_2]
masterOnly 1
[$iface_master_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
slaveOnly 0
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 248
clockAccuracy 0xFE
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
```

```
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
```



```

userDescription ;
timeSource 0xA0
recommend:
- profile: "boundary"
priority: 4
match:
- nodeLabel: "node-role.kubernetes.io/$mcp"

```

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

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4IOpts: "-2 --summary_interval -4"
      phc2sysOpts: "-r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffSet: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 100
          pins: $e810_pins
          # "$iface_master":
          # "U.FL2": "0 2"
          # "U.FL1": "0 1"
          # "SMA2": "0 2"
          # "SMA1": "0 1"
          ublxCmds:
            - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
              - "-P"
              - "29.20"
              - "-z"
              - "CFG-HW-ANT_CFG_VOLTCTRL,1"
            reportOutput: false
            - args: #ubxtool -P 29.20 -e GPS
              - "-P"
              - "29.20"
              - "-e"
              - "GPS"
            reportOutput: false
            - args: #ubxtool -P 29.20 -d Galileo
              - "-P"
              - "29.20"

```

```

    - "-d"
    - "Galileo"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d GLONASS
    - "-P"
    - "29.20"
    - "-d"
    - "GLONASS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d BeiDou
    - "-P"
    - "29.20"
    - "-d"
    - "BeiDou"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d SBAS
    - "-P"
    - "29.20"
    - "-d"
    - "SBAS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
    - "-P"
    - "29.20"
    - "-t"
    - "-w"
    - "5"
    - "-v"
    - "1"
    - "-e"
    - "SURVEYIN,600,50000"
    reportOutput: true
  - args: #ubxtool -P 29.20 -p MON-HW
    - "-P"
    - "29.20"
    - "-p"
    - "MON-HW"
    reportOutput: true
ts2phcOpts: " "
ts2phcConf: |
  [nmea]
  ts2phc.master 1
  [global]
  use_syslog 0
  verbose 1
  logging_level 7
  ts2phc.pulsewidth 100000000
  #cat /dev/GNSS to find available serial port
  #example value of gnss_serialport is /dev/ttyGNSS_1700_0
  ts2phc.nmea_serialport $gnss_serialport
  leapfile /usr/share/zoneinfo/leap-seconds.list
  [$iface_master]
  ts2phc.extts_polarity rising
  ts2phc.extts_correction 0
ptp4lConf: |
  [$iface_master]

```

```
masterOnly 1
[$iface_master_1]
masterOnly 1
[$iface_master_2]
masterOnly 1
[$iface_master_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
```

```
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0x20
recommend:
- profile: "grandmaster"
priority: 4
match:
- nodeLabel: "node-role.kubernetes.io/$mcp"
```

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

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

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

7.7.7. 拡張調整済みプロファイル

DU ワークロードを実行する単一ノードの OpenShift クラスタには、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
    [sysctl]
    kernel.timer_migration=1
    [scheduler]
    group.ice-ntp=0:f:10*:ice-ntp.*
    group.ice-gnss=0:f:10*:ice-gnss.*
    [service]
    service.stalld=start,enable
    service.chronyd=stop,disable
  recommend:
```

```
- machineConfigLabels:
  machineconfiguration.openshift.io/role: "$mcp"
  priority: 19
  profile: performance-patch
```

表7.4 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 | <ul style="list-style-type: none"> ● 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。 ● 非リアルタイムカーネルを使用する場合は、[sysctl] セクションから timer_migration override 行を削除します。 |

7.7.8. SR-IOV

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



注記

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

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

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
  annotations: {}
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: true
enableOperatorWebhook: true
logLevel: 0
```

表7.5 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 | <p>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は、コンテナ仕様の 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。</p> <p>以下に例を示します。</p> <pre>containers: - name: my-sriov-workload-container resources: limits: openshift.io/<resource_name>: "1" requests: openshift.io/<resource_name>: "1"</pre> |
| spec.enableOperatorWebhook | <p>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。</p> |

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7.6 シングルノード OpenShift クラスター用の SrioNetwork CR オプション

| SrioNetwork 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 SrioNetworkNodePolicy CR 設定 (SrioNetworkNodePolicy.yaml)

```
apiVersion: srioNetwork.openshift.io/v1
kind: SrioNetworkNodePolicy
metadata:
  name: $name
  namespace: openshift-srio-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7.7 シングルノード OpenShift クラスター用の SrioNetworkPolicy CR オプション

| SrioNetworkNodePolicy 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。 |

| SriovNetworkNodePolicy CR フィールド | 説明 |
|---------------------------------|---------------------------------|
| spec.nicSelector.pfNames | 物理機能の正確な名前は、ハードウェアと一致する必要があります。 |

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

```

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

```

7.7.9. Console Operator

クラスターケイパビリティ機能を使用して、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\"}}"
```

7.7.10. Alertmanager

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

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

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
  annotations: {}
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false

```

```
telemeterClient:
  enabled: false
prometheusK8s:
  retention: 24h
```

7.7.11. 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
```

7.7.12. LVM ストレージ

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



注記

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

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

推奨される LVMCluster 設定 (StorageLVMCluster.yaml)

```
apiVersion: lvm.topolvm.io/v1alpha1
kind: LVMCluster
metadata:
  name: odf-lvmcluster
  namespace: openshift-storage
spec:
  storage:
    deviceClasses:
      - name: vg1
    deviceSelector:
      paths:
        - /usr/disk/by-path/pci-0000:11:00.0-nvme-1
    thinPoolConfig:
```

```
name: thin-pool-1
overprovisionRatio: 10
sizePercent: 90
```

表7.8 シングルノード OpenShift クラスター用の LVMCluster CR オプション

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

7.7.13. ネットワーク診断

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

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

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

関連情報

- [ZTP を使用した遠端サイトのデプロイメント](#)

第8章 VDU アプリケーションワークロードの単一ノード OPENSIFT クラスターチューニングの検証

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

関連情報

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

8.1. VDU クラスターホストの推奨ファームウェア設定

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



注記

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

表8.1 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的高速リンクを提供します。 |
| UEFI | 有効 | vDU ホストの UEFI からの起動を有効にします。 |
| CPU パワーとパフォーマンスポリシー | パフォーマンス | CPU パワーとパフォーマンスポリシーを設定し、エネルギー効率よりもパフォーマンスを優先してシステムを最適化します。 |
| Uncore Frequency Scaling | Disabled | Uncore Frequency Scaling を無効にして、CPU のコア以外の部分の電圧と周波数が個別に設定されるのを防ぎます。 |
| Uncore Frequency | 最大 | キャッシュやメモリーコントローラーなど、CPU のコア以外の部分を可能な最大動作周波数に設定します。 |
| パフォーマンスの制限 | Disabled | プロセッサの Uncore Frequency 調整を防ぐために、パフォーマンス P 制限を無効にします。 |

| ファームウェア設定 | 設定 | 説明 |
|-------------------------------|-----------|--|
| 強化された Intel® SpeedStep テクノロジー | 有効 | Enhanced Intel SpeedStep を有効にして、システムがプロセッサ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、プロセッサコアが電力、電流、および温度の仕様制限を下回って動作している場合、自動的に定格動作周波数よりも高速に動作できるようにします。 |
| Intel Configurable TDP | 有効 | CPU の Thermal Design Power (TDP) を有効にします。 |
| 設定可能な TDP レベル | レベル 2 | TDP レベルは、特定のパフォーマンス評価に必要な CPU 消費電力を設定します。TDP レベル 2 は、消費電力を犠牲にして、CPU を最も安定したパフォーマンスレベルに設定します。 |
| energy Efficient Turbo | Disabled | Energy Efficient Turbo を無効にして、プロセッサ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、プロセッサを完全にアクティブな状態 (C0) に設定するか、ソフトウェアで実行されている CPU 内部クロックを停止します (C1)。 |
| C1E | Disabled | CPU Enhanced Halt (C1E) は、Intel チップの省電力機能です。C1E を無効にすると、非アクティブ時にオペレーティングシステムが停止コマンドを CPU に送信することを防ぎます。 |
| Processor C6 | Disabled | C6 節電は、アイドル状態の CPU コアとキャッシュを自動的に無効にする CPU 機能です。C6 を無効にすると、システムパフォーマンスが向上します。 |
| サブ NUMA クラスタリング | Disabled | サブ NUMA クラスタリングは、プロセッサコア、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。

8.2. VDU アプリケーションを実行するための推奨クラスター設定

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

8.2.1. 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/** フォルダにあります。

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

表8.2 推奨される GitOps ZTP MachineConfig CR

| MachineConfig CR | 説明 |
|---|--|
| 01-container-mount-ns-and-kubelet-conf-master.yaml | コンテナマウント namespace と Kubelet 設定を設定します。 |
| 01-container-mount-ns-and-kubelet-conf-worker.yaml | |
| 03-sctp-machine-config-master.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。 |
| 03-sctp-machine-config-worker.yaml | |
| 05-kdump-config-master.yaml | クラスターの kdump クラッシュレポートを設定します。 |
| 05-kdump-config-worker.yaml | |
| 06-kdump-master.yaml | |
| 06-kdump-worker.yaml | |
| 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 | クラスターの再起動後に rcu_expedited モードを無効にします。 |
| 08-set-rcu-normal-worker.yaml | |
| 99-crio-disable-wipe-master.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-crio-disable-wipe-worker.yaml | |

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



注記

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

関連情報

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

8.2.2. 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

8.2.3. 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。

```
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
```

```
- "efi=runtime"
- "module_blacklist=irdma"
```

2. **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。

```
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
      [sysctl]
      kernel.timer_migration=1
      [scheduler]
      group.ice-ntp=0:f:10:*:ice-ntp.*
      group.ice-gnss=0:f:10:*:ice-gnss.*
      [service]
      service.stalld=start,enable
      service.chrond=stop,disable
```

8.2.4. 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** をインストールしている。

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)
```


- リリースイメージコンテナを実行し、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。

- 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>
```

- 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
```

8.3. 推奨されるクラスター設定が適用されていることの確認

クラスターが正しい設定で実行されていることを確認できます。以下の手順では、DU アプリケーションを OpenShift Container Platform 4.15 クラスターにデプロイするために必要なさまざまな設定を確認する方法について説明します。

前提条件

- クラスターをデプロイし、vDU ワークロード用に調整している。
- OpenShift CLI (**oc**) がインストールされている。
- cluster-admin** 権限を持つユーザーとしてログインしている。

手順

- デフォルトの Operator Hub ソースが無効になっていることを確認します。以下のコマンドを実行します。

```
$ oc get operatorhub cluster -o yaml
```

出力例

```
spec:
  disableAllDefaultSources: true
```

- 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 ❶
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

- ❶ 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

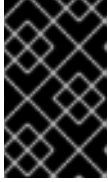
適切に設定された vDU クラスタでは、単一のアノテーション付きカタログソースのみがリスト表示されます。

- 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
...

```

- 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
```

- 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)

```

- linuxptp-daemon** コンテナ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。
 - 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.ffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      3cecef.ffe.7a7020-1
portState         SLAVE
logMinDelayReqInterval -4
peerMeanPathDelay  0
logAnnounceInterval  1
announceReceiptTimeout 3
logSyncInterval    0
delayMechanism     1
logMinPdelayReqInterval 0
versionNumber      2
3cecef.ffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      3cecef.ffe.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.ffe.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.fff.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}"
```

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}"
```

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。

- 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
```

- 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 tuned.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-ntp=0:f:10:*:ice-ntp.*
        [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
OPENSIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
```

出力例

```
Environment="OPENSIFT_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9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。

9.1. GITOPS ZTP パイプラインでの追加インストール manifests 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 manifests セットを定義できます。これらの manifests 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 リポジトリを作成している。リポジトリ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 manifests 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 manifests 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 manifests 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** コンテナから抽出し、GIT リポジトリにプッシュすることを強く推奨します。

9.2. SITECONFIG フィルターを使用したカスタムリソースのフィルタリング

フィルターを使用すると、**SiteConfig** カスタムリソース (CR) を簡単にカスタマイズして、GitOps Zero Touch Provisioning (ZTP) パイプラインのインストールフェーズで使用する他の CR を追加または除外できます。

SiteConfig CR の **inclusionDefault** 値として **include** または **exclude** を指定し、さらに、含めたり除外したりする特定の **extraManifest** RAN CR のリストを指定することもできます。**inclusionDefault** を **include** に設定すると、GitOps ZTP パイプラインはインストール中に **/source-crs/extra-manifest** 内のすべてのファイルを適用します。**inclusionDefault** を **exclude** に設定すると、その逆になります。

デフォルトで含まれている **/source-crs/extra-manifest** フォルダーから個々の CR を除外できます。以下の例では、インストール時に **/source-crs/extra-manifest/03-sctp-machine-config-worker.yaml** CR を除外するようにカスタムの単一ノード OpenShift **SiteConfig** CR を設定します。

また、いくつかのオプションのフィルタリングシナリオも説明されています。

前提条件

- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。

- カスタムサイトの設定データを管理する Git リポジトリを作成しています。リポジトリ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-4.15"
  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>" ❶
  extraManifests:
    filter:
      inclusionDefault: exclude ❷
      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
```

第10章 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。

10.1. 追加の変更のクラスターへのデプロイ

基本の GitOps Zero Touch Provisioning (ZTP) パイプライン設定以外のクラスター設定を変更する必要がある場合、次の3つのオプションを実行できます。

ZTP パイプラインの完了後に追加設定を適用する

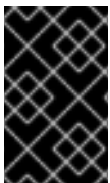
GitOps ZTP パイプラインのデプロイが完了すると、デプロイされたクラスターはアプリケーションのワークロードに対応できるようになります。この時点で、Operator を追加インストールし、お客様の要件に応じた設定を適用することができます。追加のコンフィギュレーションがプラットフォームのパフォーマンスや割り当てられた CPU バジレットに悪影響を与えないことを確認する。

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](#)

10.2. 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 がない値の更新やフィールドの挿入が可能です。

以下の手順例では、**group-du-sno-ranGen.yaml** ファイル内の **PolicyGenTemplate** CR に基づいて、参照設定用に生成された **PerformanceProfile** CR のフィールドを更新する方法について説明します。この手順を元に、**PolicyGenTemplate** の他の部分をお客様のご要望に応じて変更してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリを作成している。リポジトリ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) コンテナから抽出し、確認できます。

- a. **/out** フォルダを作成します。

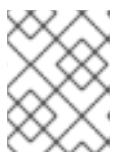
```
$ mkdir -p ./out
```

- 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:v4.15.1 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 リポジトリにプッシュします。

出力例

GitOps ZTP アプリケーションは、生成された **PerformanceProfile** CR を含む RHACM ポリシーを生成します。この CR の内容は、**PolicyGenTemplate** の **PerformanceProfile** エントリーから **metadata** と **spec** の内容をソース CR にマージすることで得られるものである。作成される CR には以下のコンテンツが含まれます。

```
---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
  cpu:
    isolated: 2-19,22-39
    reserved: 0-1,20-21
  globallyDisableIrqLoadBalancing: false
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 10
        size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
```

```

topologyPolicy: restricted
realTimeKernel:
enabled: true

```

注記

ztp-site-generate コンテナからデプロイメントした **/source-crs** フォルダでは、**\$** 構文が暗示するテンプレート置換は使用されません。むしろ、**policyGen** ツールが文字列の **\$** 接頭辞を認識し、関連する **PolicyGenTemplate** CR でそのフィールドの値を指定しない場合、そのフィールドは出力 CR から完全に省かれます。

例外として、**/source-crs** YAML ファイル内の **\$mcp** 変数は、**PolicyGenTemplate** CR から **mcp** の指定値で代用されます。例えば、**example/policygentemplates/group-du-standard-ranGen.yaml** では、**mcp** の値は **worker** となっています。

```

spec:
  bindingRules:
    group-du-standard: ""
    mcp: "worker"

```

policyGen ツールは、**\$mcp** のインスタンスを出力 CR の **worker** に置き換えます。

10.3. 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-ldp.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: <image_URL_here>
    - 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。

- 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。
- 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
```

- 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
```

10.4. POLICYGENTEMPLATE CR のポリシーコンプライアンス評価タイムアウトの設定

ハブクラスターにインストールされた Red Hat Advanced Cluster Management (RHACM) を使用して、

管理対象クラスターが適用されたポリシーに準拠しているかどうかを監視および報告します。RHACM は、ポリシーテンプレートを使用して、定義済みのポリシーコントローラーとポリシーを適用します。ポリシーコントローラーは Kubernetes のカスタムリソース定義 (CRD) インスタンスです。

デフォルトのポリシー評価間隔は、**PolicyGenTemplate** カスタムリソース (CR) でオーバーライドできます。RHACM が適用されたクラスターポリシーを再評価する前に、**ConfigurationPolicy** CR がポリシー準拠または非準拠の状態を維持できる期間を定義する期間設定を設定します。

GitOps Zero Touch Provisioning (ZTP) ポリシージェネレーターは、事前定義されたポリシー評価間隔で **ConfigurationPolicy** CR ポリシーを生成します。**noncompliant** 状態のデフォルト値は 10 秒です。**compliant** 状態のデフォルト値は 10 分です。評価間隔を無効にするには、値を **never** に設定します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- カスタムサイトの設定データを管理する Git リポジトリを作成しています。

手順

1. **PolicyGenTemplate** CR のすべてのポリシーの評価間隔を設定するには、**evaluationInterval** を **spec** フィールドに追加し、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
```

2. **PolicyGenTemplate** CR で **spec.sourceFiles** オブジェクトの評価間隔を設定するには、次の例のように、**evaluationInterval** を **sourceFiles** フィールドに追加します。

```
spec:
  sourceFiles:
    - fileName: SriovSubscription.yaml
      policyName: "sriov-sub-policy"
  evaluationInterval:
    compliant: never
    noncompliant: 10s
```

3. **PolicyGenTemplate** CR ファイルを Git リポジトリにコミットし、変更をプッシュします。

検証

マネージドスポーククラスターポリシーが予想される間隔で監視されていることを確認します。

1. 管理対象クラスターで **cluster-admin** 権限を持つユーザーとしてログインします。
2. **open-cluster-management-agent-addon** namespace で実行されている Pod を取得します。以下のコマンドを実行します。

```
$ oc get pods -n open-cluster-management-agent-addon
```

出力例

```
NAME                                READY STATUS RESTARTS   AGE
config-policy-controller-858b894c68-v4xdb  1/1   Running  22 (5d8h ago)  10d
```

3. **config-policy-controller** Pod のログで、適用されたポリシーが予想される間隔で評価されていることを確認します。

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-858b894c68-v4xdb
```

出力例

```
2022-05-10T15:10:25.280Z    info configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation due to the
policy not reaching the evaluation interval {"policy": "compute-1-config-policy-config"}
2022-05-10T15:10:25.280Z    info configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation due to the
policy not reaching the evaluation interval {"policy": "compute-1-common-compute-1-catalog-
policy-config"}
```

10.5. バリデーターインフォームポリシーを使用した GITOPS ZTP クラスターデプロイメントの完了のシグナリング

デプロイされたクラスターの GitOps Zero Touch Provisioning (ZTP) のインストールと設定が完了したときに通知するバリデーター通知ポリシーを作成します。このポリシーは、単一ノード OpenShift クラスター、3 ノードクラスター、および標準クラスターのデプロイメントに使用できます。

手順

1. ソースファイル **validatorCRs/informDuValidator.yaml** を含むスタンドアロンの **PolicyGenTemplate** カスタムリソース (CR) を作成します。スタンドアロン **PolicyGenTemplate** CR は、各クラスタータイプに1つだけ必要です。たとえば、次の CR は、単一ノードの OpenShift クラスターにバリデータ通知ポリシーを適用します。

単一ノードクラスターバリデータ通知ポリシー CR の例 (group-du-sno-validator-ranGen.yaml)

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" ❶
  namespace: "ztp-group" ❷
spec:
  bindingRules:
    group-du-sno: "" ❸
  bindingExcludedRules:
    ztp-done: "" ❹
  mcp: "master" ❺
  sourceFiles:
```

```
- fileName: validatorCRs/informDuValidator.yaml
  remediationAction: inform 6
  policyName: "du-policy" 7
```

- 1** **PolicyGenTemplates** 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、グループ **PolicyGenTemplates** 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** ファイルに存在する必要があります。
- 4** **bindingExcludedRules** で定義されたラベルは 'ztp-done:' でなければなりません。 **ztp-done** ラベルは、Topology Aware Lifecycle Manager と調整するために使用されます。
- 5** **mcp** はソースファイル **validatorCRs/informDuValidator.yaml** で使用される **MachineConfigPool** オブジェクトを定義する。これは、単一ノードの場合は **master** であり、標準のクラスターデプロイメントの場合は 3 ノードクラスターデプロイメントおよび **worker** である必要があります。
- 6** オプション: デフォルト値は **inform** です。
- 7** この値は、生成された RHACM ポリシーの名前の一部として使用されます。単一ノードの例の生成されたバリデーターポリシーは **group-du-sno-validator-du-policy** という名前です。

2. **PolicyGenTemplate** CR ファイルを Git リポジトリにコミットし、変更をプッシュします。

関連情報

- [GitOps ZTP のアップグレード](#)

10.6. POLICYGENTEMPLATES CR を使用して電源状態を設定する

低レイテンシーで高パフォーマンスのエッジデプロイメントでは、C ステートと P ステートを無効にするか制限する必要があります。この設定では、CPU は一定の周波数 (通常は最大ターボ周波数) で実行されます。これにより、CPU が常に最大速度で実行され、高いパフォーマンスと低レイテンシーが実現されます。これにより、ワークロードのレイテンシーが最適化されます。ただし、これは最大の電力消費にもつながり、すべてのワークロードに必要な可能性はありません。

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 リポジトリを作成しています。リポジトリ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の準備で説明されている手順に従っていること。

関連情報

- [ワークロードのヒントを理解する](#)
- [ワークロードヒントを手動で設定する](#)

10.6.1. PolicyGenTemplate CR を使用してパフォーマンスモードを設定する

この例に従って **group-du-sno-ranGen.yaml** の **PolicyGenTemplate** CR に基づいて、参照設定用に生成された **PerformanceProfile** CR の **workloadHints** フィールドを更新してパフォーマンスモードを設定します。

パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

1. **out/argocd/example/policygentemplates** にある **group-du-sno-ranGen.yaml** 参照ファイルの **PerformanceProfile** の **PolicyGenTemplate** エントリを次のように更新して、パフォーマンスモードを設定します。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    [...]
  spec:
    [...]
  workloadHints:
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false
```

2. Git で **PolicyGenTemplate** 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリにプッシュします。

10.6.2. PolicyGenTemplate CR を使用した高パフォーマンスモードの設定

この例に従って `group-du-sno-ranGen.yaml` の **PolicyGenTemplate** CR に基づいて、参照設定用に生成された **PerformanceProfile** CR の `workloadHints` フィールドを更新して高パフォーマンスモードを設定します。

高パフォーマンスモードは、最大の消費電力で超低遅延を提供します。

前提条件

- 低遅延および高パフォーマンスのためのホストファームウェアの設定のガイダンスに従って、パフォーマンス関連の設定で BIOS を設定しました。

手順

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

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    [...]
  spec:
    [...]
  workloadHints:
    realTime: true
    highPowerConsumption: true
    perPodPowerManagement: false
```

2. Git で **PolicyGenTemplate** 変更をコミットし、GitOps ZTP Argo CD アプリケーションによって監視される Git リポジトリにプッシュします。

10.6.3. PolicyGenTemplate CR を使用した省電力モードの設定

この例に従って `group-du-sno-ranGen.yaml` の **PolicyGenTemplate** CR に基づいて、参照設定用に生成された **PerformanceProfile** CR の `workloadHints` フィールドを更新して、省電力モードを設定します。

省電力モードは、消費電力の削減と遅延の増加のバランスをとります。

前提条件

- BIOS で C ステートと OS 制御の P ステートを有効にしました。

手順

1. `out/argocd/example/policygentemplates` にある `group-du-sno-ranGen.yaml` 参照ファイルの **PerformanceProfile** の **PolicyGenTemplate** エントリーを次のように更新して、省電力モードを設定します。追加のカーネル引数オブジェクトを使用して、省電力モード用に CPU ガバナ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" ❶
```

❶ **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

- [省電力設定のための重要なワークロードの有効化](#)
- [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の準備](#)

10.6.4. 省電力の最大化

最大の CPU 周波数を制限して、最大の電力節約を実現することを推奨します。最大 CPU 周波数を制限せずに重要でないワークロード CPU で C ステートを有効にすると、重要な CPU の周波数が高くなるため、消費電力の節約の多くが無効になります。

sysfs プラグインフィールドを更新し、リファレンス設定の **TunedPerformancePatch** CR で **max_perf_pct** に適切な値を設定することで、電力の節約を最大化します。**group-du-sno-ranGen.yaml** に基づくこの例では、最大 CPU 周波数を制限するために従う手順について説明します。

前提条件

- PolicyGenTemplate CR を使用した省電力モードの設定の説明に従って、省電力モードを設定しました。

手順

1. **out/argocd/example/policygentemplates** の **group-du-sno-ranGen.yaml** 参照ファイルで、**TunedPerformancePatch** の **PolicyGenTemplate** エントリを更新します。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 リポジトリにプッシュします。

10.7. POLICYGENTEMPLATE CR を使用した LVM ストレージの設定

GitOps Zero Touch Provisioning (ZTP) を使用して、デプロイするマネージドクラスターの論理ボリュームマネージャー (LVM) ストレージを設定できます。



注記

HTTP トランスポートで PTP イベントまたはベアメタルハードウェアイベントを使用する場合、LVM ストレージを使用してイベントサブスクリプションを永続化します。

分散ユニットでローカルボリュームを使用する永続ストレージには、Local Storage Operator を使用します。

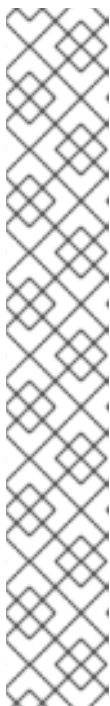
前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリを作成している。

手順

1. 新しいマネージドクラスター用に LVM ストレージを設定するには、次の YAML を **common-ranGen.yaml** ファイルの **spec.sourceFiles** に追加します。

```
- fileName: StorageLVMOSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMOSubscription.yaml
  spec:
    name: lvms-operator
    channel: stable-4.15
    policyName: subscription-policies
```



注記

Storage LVMO サブスクリプションは非推奨になりました。OpenShift Container Platform の将来のリリースでは、ストレージ LVMO サブスクリプションは利用できなくなります。代わりに、Storage LVMS サブスクリプションを使用する必要があります。

OpenShift Container Platform 4.15 では、LVMO サブスクリプションの代わりに Storage LVMS サブスクリプションを使用できます。LVMS サブスクリプションでは、**common-ranGen.yaml** ファイルを手動で上書きする必要はありません。Storage LVMS サブスクリプションを使用するには、次の YAML を **common-ranGen.yaml** ファイルの **spec.sourceFiles** に追加します。

```
- fileName: StorageLVMSubscriptionNS.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscriptionOperGroup.yaml
  policyName: subscription-policies
- fileName: StorageLVMSubscription.yaml
  policyName: subscription-policies
```

- 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" ❶
  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。シングル論理ボリュームも作成されます。

- 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にマージします。
- 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にプッシュして、GitOps ZTP を使用して LVM ストレージを新しいサイトにデプロイします。

10.8. POLICYGENTEMPLATE CR を使用した PTP イベントの設定

GitOps ZTP パイプラインを使用して、HTTP または AMQP トランスポートを使用する PTP イベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、[Red Hat AMQ Interconnect のサポートステータス](#) を参照してください。

10.8.1. 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 リポジトリを作成しています。

手順

1. 要件に応じて、以下の **PolicyGenTemplate** の変更を **group-du-3node-ranGen.yaml**、**group-du-sno-ranGen.yaml**、または **group-du-standard-ranGen.yaml** ファイルに適用してください。
 - a. **.sourceFiles** に、トランスポートホストを設定する **PtpOperatorConfig** CR ファイルを追加します。

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: http://ptp-event-publisher-service-NODE_NAME.openshift-
        ptp.svc.cluster.local:9043
```



注記

OpenShift Container Platform 4.13 以降では、PTP イベントに HTTP トランスポートを使用するときに、**PtpOperatorConfig** リソースの **transportHost** フィールドを設定する必要はありません。

- b. PTP クロックの種類とインターフェイスに **linuxptp** と **phc2sys** を設定します。たとえば、以下のスタンザを **.sourceFiles** に追加します。

```
- fileName: PtpConfigSlave.yaml ❶
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
      - name: "slave"
        interface: "ens5f1" ❷
        ptp4IOpts: "-2 -s --summary_interval -4" ❸
        phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ❹
        ptpClockThreshold: ❺
          holdOverTimeout: 30 #secs
          maxOffsetThreshold: 100 #nano secs
          minOffsetThreshold: -100 #nano secs
```

- ❶ 必要に応じて、**PtpConfigMaster.yaml**、**PtpConfigSlave.yaml**、または **PtpConfigSlaveCvl.yaml** のいずれか1つを指定できます。**PtpConfigSlaveCvl.yaml** は、Intel E810 Columbiaville NIC の **linuxptp** サービスを設定します。**group-du-sno-ranGen.yaml** および **group-du-3node-ranGen.yaml** に基づいて設定する場合は、**PtpConfigSlave.yaml** を使用します。
- ❷ デバイス固有のインターフェイス名。
- ❸ PTP 高速イベントを有効にするには、**.spec.sourceFiles.spec.profile** の **ptp4IOpts** に **--summary_interval -4** 値を追加する必要があります。
- ❹ **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。

- 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にマージします。
- 変更をサイト設定リポジトリにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- [PolicyGenTemplate CR を使用して、ソース CR の内容を上書きする。](#)

10.8.2. AMQP トランスポートを使用する PTP イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイするマネージドクラスター上で、AMQP トランスポートを使用する PTP イベントを設定できます。



注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、[Red Hat AMQ Interconnect のサポートステータス](#) を参照してください。

前提条件

- OpenShift CLI (`oc`) がインストールされている。
- `cluster-admin` 権限を持つユーザーとしてログインしている。
- カスタムサイトの設定データを管理する Git リポジトリを作成しています。

手順

- `common-ranGen.yaml` ファイルの `.spec.sourceFiles` に以下の YAML を追加し、AMQP Operator を設定します。

```
#AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
```



```

policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"

```

2. 要件に応じて、以下の **PolicyGenTemplate** の変更を **group-du-3node-ranGen.yaml**、**group-du-sno-ranGen.yaml**、または **group-du-standard-ranGen.yaml** ファイルに適用してください。

- a. **.sourceFiles** に、AMQ トランスポートホストを設定する **PtpOperatorConfig** CR ファイルを **config-policy** に追加します。

```

- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
  spec:
    daemonNodeSelector: {}
    ptpEventConfig:
      enableEventPublisher: true
      transportHost: "amqp://amq-router.amq-router.svc.cluster.local"

```

- b. PTP クロックの種類とインターフェイスに **linuxptp** と **phc2sys** を設定します。たとえば、以下のスタanzas を **.sourceFiles** に追加します。

```

- fileName: PtpConfigSlave.yaml ❶
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
      - name: "slave"
        interface: "ens5f1" ❷
        ptp4lOpts: "-2 -s --summary_interval -4" ❸
        phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ❹
        ptpClockThreshold: ❺
        holdOverTimeout: 30 #secs
        maxOffsetThreshold: 100 #nano secs
        minOffsetThreshold: -100 #nano secs

```

- ❶ 要件に応じて、**PtpConfigMaster.yaml**、**PtpConfigSlave.yaml**、または **PtpConfigSlaveCvl.yaml** を1つ指定できます。**PtpConfigSlaveCvl.yaml** は、Intel E810 Columbiaville NIC の **linuxptp** サービスを設定します。**group-du-sno-ranGen.yaml** および **group-du-3node-ranGen.yaml** に基づいて設定する場合は、**PtpConfigSlave.yaml** を使用します。

- ❷ デバイス固有のインターフェイス名。

- ❸ 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。

- ❹ **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。

- ❺ オプション: **ptpClockThreshold** スタanzas 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anzas は、デフォルトの **ptpClockThreshold** 値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定しま

す。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME (phc2sys)** またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。

3. 以下の **PolicyGenTemplate** の変更を、特定のサイトの YAML ファイル (例: **example-sno-site.yaml**) に適用してください。
 - a. **.sourceFiles** に、AMQ ルーターを設定する **Interconnect** CR ファイルを **config-policy** に追加します。

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

4. 必要なその他の変更およびファイルをカスタムサイトリポジトリにマージします。
5. 変更をサイト設定リポジトリにプッシュし、GitOps ZTP を使用して PTP 高速イベントを新規サイトにデプロイします。

関連情報

- [AMQ メッセージングバスのインストール](#)
- コンテナイメージレジストリーの詳細は、[OpenShift イメージレジストリーの概要](#) を参照してください。

10.9. 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。

10.9.1. 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 リポジトリを作成しています。

手順

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

```
# Bare Metal Event Relay operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. たとえば、**group-du-sno-ranGen.yaml** ファイルの特定のグループ設定ファイルで、**HardwareEvent** CR を **spec.sourceFiles** に追加します。

```
- fileName: HardwareEvent.yaml 1
  policyName: "config-policy"
  spec:
    nodeSelector: {}
    transportHost: "http://hw-event-publisher-service.openshift-bare-metal-
events.svc.cluster.local:9043"
    logLevel: "info"
```

- 1** 各ベースボード管理コントローラー (BMC) では、1つの **HardwareEvent** CR のみ必要です。



注記

OpenShift Container Platform 4.13 以降では、ベアメタルイベントで HTTP トランスポートを使用する場合、**HardwareEvent** カスタムリソース (CR) の **TransportHost** フィールドを設定する必要はありません。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリにマージします。
4. 変更をサイト設定リポジトリにプッシュし、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](#)

10.9.2. 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 リポジトリを作成しています。

手順

1. AMQ Interconnect Operator と Bare Metal Event Relay Operator を設定するには、次の YAML を **common-ranGen.yaml** ファイルの **spec.sourceFiles** に追加します。

```
# AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
# Bare Metal Event Rely operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. **Interconnect** CR をサイト設定ファイルの **.spec.sourceFiles** (**example-sno-site.yaml** ファイルなど) に追加します。

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

3. たとえば、**group-du-sno-ranGen.yaml** ファイルの特定のグループ設定ファイルで、**HardwareEvent** CR を **spec.sourceFiles** に追加します。

```
- fileName: HardwareEvent.yaml
  policyName: "config-policy"
  spec:
    nodeSelector: {}
```

```
transportHost: "amqp://<amq_interconnect_name>.  
<amq_interconnect_namespace>.svc.cluster.local" ❶  
logLevel: "info"
```

- ❶ **transportHost** URL は、既存の AMQ Interconnect CR **name** と **namespace** で設定されます。たとえば、**transportHost: "amqp://amq-router.amq-router.svc.cluster.local"** では、AMQ Interconnect の **name** と **namespace** の両方が **amq-router** に設定されます。



注記

各ベースボード管理コントローラー (BMC) には、単一の **HardwareEvent** リソースのみが必要です。

- 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。
- 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>"
```

10.10. 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** ディレクトリを消去するリスクがあります。イメージのダウンロード時間が長い場合の対処方法として、GitOps Zero Touch Provisioning (ZTP) を使用してリモートマネージドクラスター上にローカルイメージレジストリーを作成できます。これは、クラスターがネットワークの遠端にデプロイメントされるエッジコンピューティングシナリオで役立ちます。

GitOps ZTP を使用してローカルイメージレジストリーをセットアップする前に、リモートマネージドクラスターのインストールに使用する **SiteConfig** CR でディスクパーティショニングを設定する必要があります。インストール後、**PolicyGenTemplate** CR を使用してローカルイメージレジストリーを設定します。次に、GitOps ZTP パイプラインは永続ボリューム (PV) と永続ボリューム要求 (PVC) CR を作成し、**imageregistry** 設定にパッチを適用します。



注記

ローカルイメージレジストリーは、ユーザーアプリケーションイメージにのみ使用でき、OpenShift Container Platform または Operator Lifecycle Manager Operator イメージには使用できません。

関連情報

- [OpenShift Container Platform レジストリーの概要](#)

10.10.1. 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。



注記

再起動のたびに `/dev/sda` や `/dev/sdb` などのデバイス名が切り替わらないように、デバイスに永続的な名前を付けます。**rootDeviceHints** を使用して起動可能なデバイスを選択し、同じデバイスを使用してさらにパーティショニングすることができます。

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。
- 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 リポジトリを作成している。

手順

1. ホストディスクのパーティショニングを記述する次の YAML を、マネージドクラスターのインストールに使用する **SiteConfig** CR に追加します。

```
nodes:
  rootDeviceHints:
    wwn: "0x62cea7f05c98c2002708a0a22ff480ea"
  diskPartition:
    - device: /dev/disk/by-id/wwn-0x62cea7f05c98c2002708a0a22ff480ea ①
    partitions:
      - mount_point: /var/imageregistry
        size: 102500 ②
        start: 344844 ③
```

- ① この設定は、ハードウェアによって異なります。設定には、シリアル番号またはデバイス名を指定できます。この値は、**rootDeviceHints** に設定された値と一致する必要があります。
- ② **size** の最小値は 102500 MiB です。
- ③ **start** の最小値は 25000 MiB です。**size** と **start** の合計値がディスクサイズを超えてはなりません。超えると、インストールが失敗します。

2. **SiteConfig** CR を保存し、サイト設定リポジトリにプッシュします。

ZTP パイプラインは、**SiteConfig** CR を使用してクラスターをプロビジョニングし、ディスクパーティションを設定します。

10.10.2. PolicyGenTemplate CR を使用してイメージレジストリーを設定する

PolicyGenTemplate (PGT) CR を使用して、イメージレジストリーの設定に必要な CR を適用し、**imageregistry** 設定にパッチを適用します。

前提条件

- 管理対象クラスターでディスクパーティションを設定しました。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- GitOps Zero Touch Provisioning (ZTP) で使用するカスタムサイト設定データを管理する Git リポジトリを作成している。

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 リポジトリにプッシュします。

検証

次の手順を使用して、マネージドクラスターのローカルイメージレジストリーに関するエラーをトラブルシューティングします。

- マネージドクラスターにログインしているときに、レジストリーへのログインが成功したことを確認します。以下のコマンドを実行します。

- a. マネージドクラスター名をエクスポートします。

```
$ cluster=<managed_cluster_name>
```

- b. マネージドクラスター **kubeconfig** の詳細を取得します。

```
$ oc get secret -n $cluster $cluster-admin-password -o jsonpath='{.data.password}' |
base64 -d > kubeadmin-password-$cluster
```

- c. クラスター **kubeconfig** をダウンロードしてエクスポートします。

```
$ oc get secret -n $cluster $cluster-admin-kubeconfig -o jsonpath='{.data.kubeconfig}' |
base64 -d > kubeconfig-$cluster && export KUBECONFIG=./kubeconfig-$cluster
```

- d. マネージドクラスターからイメージレジストリーへのアクセスを確認します。レジストリーへのアクセスを参照してください。

- **imageregistry.operator.openshift.io** グループインスタンスの **Config** CRD がエラーを報告していないことを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get image.config.openshift.io cluster -o yaml
```

出力例


```

apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2021-10-08T19:02:39Z"
  generation: 5
  name: cluster
  resourceVersion: "688678648"
  uid: 0406521b-39c0-4cda-ba75-873697da75a4
spec:
  additionalTrustedCA:
    name: acm-ice

```

- 管理対象クラスターの **PersistentVolumeClaim** にデータが入力されていることを確認します。マネージドクラスターにログインしているときに、次のコマンドを実行します。

```
$ oc get pv image-registry-sc
```

- **registry*** Pod が実行中であり、**openshift-image-registry** namespace にあることを確認します。

```
$ oc get pods -n openshift-image-registry | grep registry*
```

出力例

```

cluster-image-registry-operator-68f5c9c589-42cfg 1/1 Running 0 8d
image-registry-5f8987879-6nx6h 1/1 Running 0 8d

```

- マネージドクラスターのディスクパーティションが正しいことを確認します。
 - a. マネージドクラスターへのデバッグシェルを開きます。

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

- b. **lsblk** を実行して、ホストディスクパーティションを確認します。

```

sh-4.4# lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 446.6G 0 disk
|-sda1 8:1 0 1M 0 part
|-sda2 8:2 0 127M 0 part
|-sda3 8:3 0 384M 0 part /boot
|-sda4 8:4 0 336.3G 0 part /sysroot
`-sda5 8:5 0 100.1G 0 part /var/imageregistry ❶
sdb 8:16 0 446.6G 0 disk
sr0 11:0 1 104M 0 rom

```

- ❶ **/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](#)

10.11. POLICYGENTEMPLATE CR でのハブテンプレートの使用

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

ハブ側のクラスターテンプレートを使用すると、ターゲットクラスターに合わせて動的にカスタマイズできる設定ポリシーを定義できます。これにより、設定は似ているが値が異なる多くのクラスターに対して個別のポリシーを作成する必要がなくなります。



重要

ポリシーテンプレートは、ポリシーが定義されている namespace と同じ namespace に制限されています。これは、ハブテンプレートで参照されるオブジェクトを、ポリシーが作成されたのと同じ namespace に作成する必要があることを意味します。

TALM を使用する GitOps ZTP では、次のサポートされているハブテンプレート関数を使用できます。

- **fromConfigmap** は、指定された **ConfigMap** リソースで提供されたデータキーの値を返します。



注記

ConfigMap CR には **1 MiB のサイズ制限** があります。**ConfigMap** CR の有効サイズは、**last-applied-configuration** アノテーションによってさらに制限されます。**last-applied-configuration** 制限を回避するには、次のアノテーションをテンプレート **ConfigMap** に追加します。

```
argocd.argoproj.io/sync-options: Replace=true
```

- **base64enc** は、base64 でエンコードされた入力文字列の値を返します
- **base64dec** は、base64 でエンコードされた入力文字列のデコードされた値を返します
- **indent** は、インデントスペースが追加された入力文字列を返します
- **autoindent** は、親テンプレートで使用されているスペースに基づいてインデントスペースを追加した入力文字列を返します。
- **toInt** は、入力値の整数値をキャストして返します
- **toBool** は入力文字列をブール値に変換し、ブール値を返します

GitOps ZTP では、さまざまな [オープンソースコミュニティ機能](#) も利用できます。

関連情報

- [設定ポリシーでのハブクラスターテンプレートの RHACM サポート](#)

10.11.1. ハブテンプレートの例

次のコード例は、有効なハブテンプレートです。これらの各テンプレートは、**default** namespace で **test-config** という名前の **ConfigMap** CR から値を返します。

- キー **common-key** を持つ値を返します。

```
{{hub fromConfigMap "default" "test-config" "common-key" hub}}
```

- **.ManagedClusterName** フィールドと文字列 **-name** の連結値を使用して、文字列を返します。

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) hub}}
```

- **.ManagedClusterName** フィールドと文字列 **-name** の連結値からブール値をキャストして返します。

```
{{hub fromConfigMap "default" "test-config" (printf "%s-name" .ManagedClusterName) | toBool hub}}
```

- **.ManagedClusterName** フィールドと文字列 **-name** の連結値から整数値をキャストして返します。

```
{{hub (printf "%s-name" .ManagedClusterName) | fromConfigMap "default" "test-config" | toInt hub}}
```

10.11.2. ハブテンプレートを使用して PolicyGenTemplate CR グループのグループとサイトの設定を指定する

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。**PolicyGenTemplate** (PGT) 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 **PolicyGenTemplate** CR を作成する必要がなくなります。

ハードウェアの種類や地域などのユースケースに応じて、フリート内のクラスターをさまざまなカテゴリーにグループ化できます。各クラスターには、そのクラスターが属するグループ (複数可) に対応するラベルが必要です。各グループの設定値を異なる **ConfigMap** CR で管理する場合、ハブテンプレートを使用してグループ内のすべてのクラスターに変更を適用するには、1つのグループ **PolicyGenTemplate** CR のみ必要です。

次の例は、3つの **ConfigMap** CR と1つの **PolicyGenTemplate** CR グループを使用して、サイトとグループの両方の設定を、ハードウェアタイプとリージョンごとにグループ化されたクラスターに適用する方法を示しています。



注記

fromConfigmap 関数を使用する場合、**printf** 変数はテンプレートリソース **data** キーフィールドでのみ使用できます。**name** および **namespace** フィールドでは使用できません。

前提条件

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

- カスタムサイトの設定データを管理する Git リポジトリを作成しています。リポジトリ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 ❶
data:
  # SrioNetworkNodePolicy.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"

```

- ❶ **argocd.argoproj.io/sync-options** アノテーションは、**ConfigMap** のサイズが 1 MiB より大きい場合にのみ必要です。

- b. **group-zones-configmap** という名前の **ConfigMap** CR を作成して、地域設定を保持します。以下に例を示します。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: group-zones-configmap
  namespace: ztp-group
data:
  # ClusterLogForwarder.yaml
  zone-1-cluster-log-fwd-outputs: "[{\"type\":\"kafka\", \"name\":\"kafka-open\", \"url\":\"tcp://10.46.55.190:9092/test\"}]"
  zone-1-cluster-log-fwd-pipelines: "[{\"inputRefs\": [\"audit\", \"infrastructure\"], \"labels\": {\"label1\": \"test1\", \"label2\": \"test2\", \"label3\": \"test3\", \"label4\": \"test4\"}, \"name\": \"all-to-default\", \"outputRefs\": [\"kafka-open\"]}]"

```

- c. **site-data-configmap** という名前の **ConfigMap** CR を作成して、サイト固有の設定を保持します。以下に例を示します。

```

apiVersion: v1
kind: ConfigMap
metadata:

```

```

name: site-data-configmap
namespace: ztp-group
data:
  # SriovNetwork.yaml
  du-sno-1-zone-1-sriov-network-vlan-1: "140"
  du-sno-1-zone-1-sriov-network-vlan-2: "150"

```



注記

各 **ConfigMap** CR は、**PolicyGenTemplate** CR グループから生成されるポリシーと同じ namespace に配置される必要があります。

2. Git で **ConfigMap** CR をコミットし、Argo CD アプリケーションが監視する Git リポジトリにプッシュします。
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. 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 **PolicyGenTemplate** CR グループを作成します。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プレートコンテキスト値です。

5. サイトの **PolicyGenTemplate** CR を Git にコミットし、ArgoCD アプリケーションによって監視されている Git リポジトリにプッシュします。

注記

参照された **ConfigMap** CR に対するその後の変更は、適用されたポリシーに自動的に同期されません。新しい **ConfigMap** の変更を手動で同期して、既存の **PolicyGenTemplate** CR を更新する必要があります。「新しい ConfigMap の変更を既存の 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 **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。

10.11.3. 新しい ConfigMap の変更を既存の 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 **PolicyGenTemplate** CR を作成しました。

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 **PolicyGenTemplate** CR を使用して、削除されたポリシーをすぐに再作成します。たとえば、以下のコマンドを実行します。

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

- b. オプション 2: **ConfigMap** を更新するたびに、特別なアノテーション **policy.open-cluster-management.io/trigger-update** を異なる値でポリシーに適用します。以下に例を示します。

```
$ oc annotate policy <policy_name> -n <policy_namespace> policy.open-cluster-management.io/trigger-update="1"
```



注記

変更を有効にするには、更新されたポリシーを適用する必要があります。詳細については、[再処理のための特別なアノテーション](#) を参照してください。

3. オプション: 存在する場合は、ポリシーを含む **ClusterGroupUpgrade** 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 **ClusterGroupUpgrade** 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11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。

11.1. TOPOLOGY AWARE LIFECYCLE MANAGER の設定について

Topology Aware Lifecycle Manager (TALM) は、1つまたは複数の OpenShift Container Platform クラスターに対する Red Hat Advanced Cluster Management (RHACM) ポリシーのデプロイメントを管理します。TALM を大規模なクラスターのネットワークで使用することにより、限られたバッチで段階的にポリシーをクラスターにデプロイメントすることができます。これにより、更新時のサービス中断の可能性を最小限に抑えることができます。TALM では、以下の動作を制御することができます。

- 更新のタイミング
- RHACM マネージドクラスター数
- ポリシーを適用するマネージドクラスターのサブセット
- クラスターの更新順序
- クラスターに修正されたポリシーのセット
- クラスターに修正されるポリシーの順序
- カナリアクラスターの割り当て

シングルノードの OpenShift の場合、Topology Aware Lifecycle Manager (TALM) は次の機能を提供します。

- アップグレード前に、デプロイメントのバックアップを作成する
- 帯域幅が制限されたクラスターのイメージの事前キャッシュ

TALM は、OpenShift Container Platform y-stream および z-stream 更新のオーケストレーションをサポートし、y-streams および z-streams での day-two 操作をサポートします。

11.2. 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
- **PolicyGenTemplate** カスタムリソース定義 (CRD) から自動生成されたポリシー

手動承認で Operator 契約を更新するポリシーのために、TALM は、更新された Operator のインストールを承認する追加機能を提供します。

マネージドポリシーの詳細は、RHACM のドキュメントの [ポリシーの概要](#) を参照してください。

PolicyGenTemplate CRD の詳細は、「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」の「PolicyGenTemplate CRD について」のセクションを参照してください。

11.3. WEB コンソールを使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール

OpenShift Container Platform Web コンソールを使用して Topology Aware Lifecycle Manager をインストールできます。

前提条件

- 最新バージョンの RHACM Operator をインストールします。
- 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. 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 のコンテナのログを確認します。

11.4. CLI を使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール

OpenShift CLI (**oc**) を使用して Topology Aware Lifecycle Manager (TALM) をインストールできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 最新バージョンの RHACM Operator をインストールします。
- 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.4.15.x  Topology Aware Lifecycle Manager  4.15.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                                cluster-group-upgrades-controller-manager  1/1
1      1      14s
```

11.5. CLUSTERGROUPUPGRADE CR

Topology Aware Lifecycle Manager (TALM) は、クラスター グループの **ClusterGroupUpgrade** CR から修復計画を作成します。**ClusterGroupUpgrade** CR で次の仕様を定義できます。

- グループのクラスター
- **ClusterGroupUpgrade** CR のブロック
- 管理ポリシーの適用リスト
- 同時更新の数
- 適用可能なカナリア更新
- 更新前後に実行するアクション
- 更新タイミング

ClusterGroupUpgrade CR の **enable** フィールドを使用して、更新の開始時刻を制御できます。たとえば、メンテナンスウィンドウが 4 時間にスケジュールされている場合、**enable** フィールドを **false** に設定して **ClusterGroupUpgrade** CR を準備できます。

次のように **spec.remediationStrategy.timeout** 設定を設定することで、タイムアウトを設定できます。

```
spec
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

batchTimeoutAction を使用して、クラスターの更新が失敗した場合にどうなるかを判断できます。**continue** を指定して失敗したクラスターをスキップし、他のクラスターのアップグレードを続行するか、**abort** を指定してすべてのクラスターのポリシー修正を停止することができます。タイムアウトが経過すると、TALM はすべての **enforce** ポリシーを削除して、クラスターがそれ以上更新されないようにします。

変更を適用するには、**enabled** フィールドを **true** に設定します。

詳細については、管理対象クラスターへの更新ポリシーの適用セクションを参照してください。

TALM は指定されたクラスターへのポリシーの修復を通じて機能するため、**ClusterGroupUpgrade** CR は多くの条件について true または false のステータスを報告できます。



注記

TALM がクラスターの更新を完了した後、同じ **ClusterGroupUpgrade** CR の制御下でクラスターが再度更新されることはありません。次の場合は、新しい **ClusterGroupUpgrade** CR を作成する必要があります。

- クラスターを再度更新する必要がある場合
- クラスターが更新後に **inform** ポリシーで非準拠に変更された場合

11.5.1. 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クターのリストで設定されます。リスト内の各セクターは、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: ❶
    addClusterLabels:
      upgrade-done: ""
    deleteClusterLabels:
      upgrade-running: ""
    deleteObjects: true
    beforeEnable: ❷
    addClusterLabels:
      upgrade-running: ""
  backup: false
  clusters: ❸
    - spoke1
  enable: false ❹
  managedPolicies: ❺
    - talm-policy
  preCaching: false
```

```

remediationStrategy: 6
  canaries: 7
    - spoke1
  maxConcurrency: 2 8
  timeout: 240
clusterLabelSelectors: 9
  - matchExpressions:
    - key: label1
      operator: In
      values:
        - value1a
        - value1b
batchTimeoutAction: 10
status: 11
  computedMaxConcurrency: 2
  conditions:
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: All selected clusters are valid
      reason: ClusterSelectionCompleted
      status: 'True'
      type: ClustersSelected 12
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: Completed validation
      reason: ValidationCompleted
      status: 'True'
      type: Validated 13
    - lastTransitionTime: '2022-11-18T16:37:16Z'
      message: Not enabled
      reason: NotEnabled
      status: 'False'
      type: Progressing
  managedPoliciesForUpgrade:
    - name: talm-policy
      namespace: talm-namespace
  managedPoliciesNs:
    talm-policy: talm-namespace
  remediationPlan:
    - - spoke1
      - - spoke2
        - spoke3
  status:

```

- 1 各クラスターのポリシー修正が完了したときに TALM が実行するアクションを指定します。
- 2 更新プロセスを開始するときに TALM が実行するアクションを指定します。
- 3 更新するクラスターの一覧を定義します。
- 4 **enable** フィールドは **false** に設定されています。
- 5 修正するユーザー定義のポリシーセットを一覧表示します。
- 6 クラスター更新の詳細を定義します。
- 7 カナリア更新のクラスターを定義します。

- 8 バッチの同時更新の最大数を定義します。修復バッチの数は、カナリアクラスターの数に加えて、カナリアクラスターを除くクラスターの数 **maxConcurrency** 値で除算します。すべての管理ポ
- 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のみ、**spec** フィールドを変更できます。

11.5.2. 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せん。

11.5.3. 事前キャッシュ

クラスターにはコンテナイメージレジストリーにアクセスするための帯域幅が制限されるため、更新が完了する前にタイムアウトが発生する可能性があります。シングルノードの OpenShift クラスターでは、事前キャッシュを使用して、これを回避できます。**preCaching** フィールドを **true** に設定して **ClusterGroupUpgrade** CR を作成すると、コンテナイメージの事前キャッシュが開始されます。TALM は、使用可能なディスク容量を OpenShift Container Platform イメージの推定サイズと比較して、十分な容量があることを確認します。クラスターに十分なスペースがない場合、TALM はそのクラスターの事前キャッシュをキャンセルし、そのクラスターのポリシーを修復しません。

TALM は **PrecacheSpecValid** 条件を使用して、次のようにステータス情報を報告します。

- **true**
事前キャッシュの仕様は有効で一貫性があります。
- **false**

事前キャッシュの仕様は不完全です。

TALM は **PrecachingSucceeded** 条件を使用して、次のようにステータス情報を報告します。

- **true**
TALM は事前キャッシュプロセスを完了しました。いずれかのクラスターで事前キャッシュが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。クラスターの事前キャッシュが失敗した場合は、メッセージで通知されます。
- **false**
1つ以上のクラスターで事前キャッシュがまだ進行中か、すべてのクラスターで失敗しました。

詳細については、コンテナ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。

11.5.4. バックアップの作成

単一ノードの OpenShift の場合、TALM は更新前にデプロイメントのバックアップを作成できます。アップデートが失敗した場合は、以前のバージョンを回復し、アプリケーションの再プロビジョニングを必要とせずにクラスターを動作状態に復元できます。バックアップ機能を使用するには、最初に **backup** フィールドを **true** に設定して **ClusterGroupUpgrade** CR を作成します。バックアップの内容が最新であることを確認するために、**ClusterGroupUpgrade** CR の **enable** フィールドを **true** に設定するまで、バックアップは取得されません。

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

- **true**
すべてのクラスターのバックアップが完了したか、バックアップの実行が完了したが、1つ以上のクラスターで失敗しました。いずれかのクラスターのバックアップが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。
- **false**
1つ以上のクラスターのバックアップがまだ進行中か、すべてのクラスターのバックアップが失敗しました。

詳細については、アップグレード前のクラスターリソースのバックアップの作成セクションを参照してください。

11.5.5. クラスターの更新

TALM は、修復計画に従ってポリシーを適用します。以降のバッチに対するポリシーの適用は、現在のバッチのすべてのクラスターがすべての管理ポリシーに準拠した直後に開始されます。バッチがタイムアウトすると、TALM は次のバッチに移動します。バッチのタイムアウト値は、**spec.timeout** フィールドは修復計画のバッチ数で除算されます。

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

- **true**
TALM は準拠していないポリシーを修正しています。
- **false**
更新は進行中ではありません。これには次の理由が考えられます。
 - すべてのクラスターは、すべての管理ポリシーに準拠しています。
 - ポリシーの修復に時間がかかりすぎたため、更新がタイムアウトしました。

- ブロッキング CR がシステムにないか、まだ完了していません。
- **ClusterGroupUpgrade** CR が有効になっていません。
- バックアップはまだ進行中です。



注記

管理されたポリシーは、**ClusterGroupUpgrade** CR の **managedPolicies** フィールドに一覧表示される順序で適用されます。1つの管理ポリシーが一度に指定されたクラスターに適用されます。クラスターが現在のポリシーに準拠している場合、次の管理ポリシーがクラスターに適用されます。

Progressing 状態の ClusterGroupUpgrade CR の例

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: true
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 2
    timeout: 240
  clusterLabelSelectors:
    - matchExpressions:
      - key: label1
        operator: In
        values:
          - value1a
          - value1b
  batchTimeoutAction:
status:
  clusters:
    - name: spoke1
      state: complete

```

```

computedMaxConcurrency: 2
conditions:
- lastTransitionTime: '2022-11-18T16:27:15Z'
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: 'True'
  type: ClustersSelected
- lastTransitionTime: '2022-11-18T16:27:15Z'
  message: Completed validation
  reason: ValidationCompleted
  status: 'True'
  type: Validated
- lastTransitionTime: '2022-11-18T16:37:16Z'
  message: Remediating non-compliant policies
  reason: InProgress
  status: 'True'
  type: Progressing ①
managedPoliciesForUpgrade:
- name: talm-policy
  namespace: talm-namespace
managedPoliciesNs:
  talm-policy: talm-namespace
remediationPlan:
- - spoke1
  - spoke2
  - spoke3
status:
  currentBatch: 2
  currentBatchRemediationProgress:
    spoke2:
      state: Completed
    spoke3:
      policyIndex: 0
      state: InProgress
  currentBatchStartedAt: '2022-11-18T16:27:16Z'
  startedAt: '2022-11-18T16:27:15Z'

```

① **Progressing** フィールドは、TALM がポリシーの修復中であることを示しています。

11.5.6. 更新ステータス

TALM は **Succeeded** 条件を使用して、ステータスと理由を次のようにレポートします。

- **true**
すべてのクラスターは、指定された管理ポリシーに準拠しています。
- **false**
修正に使用できるクラスターがないか、次のいずれかの理由でポリシーの修正に時間がかかりすぎたため、ポリシーの修正に失敗しました。
 - 現在のバッチにカナリア更新が含まれており、バッチ内のクラスターがバッチタイムアウト内のすべての管理ポリシーに準拠していません。
 - クラスターは、**remediationStrategy** フィールドに指定された **timeout** 値内で管理ポリシーに準拠していませんでした。

Succeeded 状態の ClusterGroupUpgrade CR の例

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
    - spoke1
    - spoke4
  enable: true
  managedPolicies:
    - policy1-common-cluster-version-policy
    - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: ❶
  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 ❷
    - message: All clusters are compliant with all the managed policies
      reason: Completed
      status: "True"
      type: Succeeded ❸
  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'

```

❷ **Progressing** フィールドでは、更新が完了したため、ステータスは **false** です。クラスターはすべての管理ポリシーに準拠しています。

- 3 **Succeeded** フィールドは、検証が正常に完了したことを示しています。
- 1 **status** フィールドには、クラスターのリストとそれぞれのステータスが含まれます。クラスターのステータスは、**complete** または **timedout** です。

タイムアウト 状態の ClusterGroupUpgrade CR の例

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
    - spoke1
    - spoke2
  enable: true
  managedPolicies:
    - talm-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
status:
  clusters:
    - name: spoke1
      state: complete
    - currentPolicy: 1
      name: talm-policy
      status: NonCompliant
      name: spoke2
      state: timedout
  computedMaxConcurrency: 2
  conditions:
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: All selected clusters are valid
      reason: ClusterSelectionCompleted
      status: 'True'
      type: ClustersSelected
    - lastTransitionTime: '2022-11-18T16:27:15Z'
      message: Completed validation
      reason: ValidationCompleted
      status: 'True'
      type: Validated

```

```

- lastTransitionTime: '2022-11-18T16:37:16Z'
  message: Policy remediation took too long
  reason: TimedOut
  status: 'False'
  type: Progressing
- lastTransitionTime: '2022-11-18T16:37:16Z'
  message: Policy remediation took too long
  reason: TimedOut
  status: 'False'
  type: Succeeded ②
managedPoliciesForUpgrade:
- name: talm-policy
  namespace: talm-namespace
managedPoliciesNs:
  talm-policy: talm-namespace
remediationPlan:
- - spoke1
  - spoke2
status:
  startedAt: '2022-11-18T16:27:15Z'
  completedAt: '2022-11-18T20:27:15Z'

```

- ① クラスターの状態が **timedout** の場合、**currentPolicy** フィールドにはポリシーの名前とポリシーのステータスが表示されます。
- ② **succeeded** のステータスは **false** であり、ポリシーの修正に時間がかかりすぎたことを示すメッセージが表示されます。

11.5.7. ClusterGroupUpgrade CR のブロック

複数の **ClusterGroupUpgrade** CR を作成して、それらの適用順序を制御できます。

たとえば、**ClusterGroupUpgrade** CR A の開始をブロックする **ClusterGroupUpgrade** CR C を作成する場合、**ClusterGroupUpgrade** CR A は **ClusterGroupUpgrade** CR C のステータスが **UpgradeComplete** になるまで起動できません。

1つの **ClusterGroupUpgrade** CR には複数のブロッキング CR を含めることができます。この場合、現在の CR のアップグレードを開始する前に、すべてのブロッキング CR を完了する必要があります。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1つ以上のマネージドクラスターをプロビジョニングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. **ClusterGroupUpgrade** CR の内容を **cgu-a.yaml**、**cgu-b.yaml**、および **cgu-c.yaml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ntp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ntp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-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-ntp-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
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ntp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ntp-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-ntp-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-ntp-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-ntp-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
    copiedPolicies:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    managedPoliciesCompliantBeforeUpgrade:
    - policy2-common-pao-sub-policy
    - policy3-common-ntp-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-ntp-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]' ❶
      reason: UpgradeCannotStart
      status: "False"
      type: Ready
  copiedPolicies:
    - cgu-a-policy1-common-cluster-version-policy
    - cgu-a-policy2-common-pao-sub-policy
    - cgu-a-policy3-common-ntp-sub-policy
  managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy2-common-pao-sub-policy
      namespace: default
    - name: policy3-common-ntp-sub-policy
      namespace: default
  placementBindings:
    - cgu-a-policy1-common-cluster-version-policy
    - cgu-a-policy2-common-pao-sub-policy
    - cgu-a-policy3-common-ntp-sub-policy
```

```

placementRules:
- cgu-a-policy1-common-cluster-version-policy
- cgu-a-policy2-common-pao-sub-policy
- cgu-a-policy3-common-ntp-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-ntp-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
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ntp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ntp-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-ntp-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-ntp-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-ntp-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
copiedPolicies:
- cgu-c-policy1-common-cluster-version-policy
- cgu-c-policy4-common-sriov-sub-policy
managedPoliciesCompliantBeforeUpgrade:
- policy2-common-pao-sub-policy
- policy3-common-ntp-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。

11.6. マネージドクラスターでのポリシーの更新

Topology Aware Lifecycle Manager (TALM) は、**ClusterGroupUpgrade** CR で指定されたクラスターの **inform** ポリシーのセットを修正します。TALM は、マネージドの RHACM ポリシーの **enforce** コピーを作成することにより、**inform** ポリシーを修正します。コピーされた各ポリシーには、それぞれの対応する RHACM 配置ルールと RHACM 配置バインディングがあります。

1つずつ、TALM は、現在のバッチから、適用可能な管理ポリシーに対応する配置ルールに各クラスターを追加します。クラスターがポリシーにすでに準拠している場合は、TALM は準拠するクラスターへのポリシーの適用を省略します。次に TALM は次のポリシーを非準拠クラスターに適用します。TALM がバッチの更新を完了すると、コピーしたポリシーに関連付けられた配置ルールからすべてのクラスターが削除されます。次に、次のバッチの更新が開始されます。

スポーククラスターの状態が RHACM に準拠している状態を報告しない場合、ハブクラスターの管理ポリシーには TALM が必要とするステータス情報がありません。TALM は、以下の方法でこれらのケースを処理します。

- ポリシーの **status.compliant** フィールドがない場合、TALM はポリシーを無視してログエントリを追加します。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** がいない場合、TALM はエラーを生成します。
- クラスターのコンプライアンスステータスがポリシーの **status.status** フィールドにない場合、TALM はそのクラスターをそのポリシーに準拠していないと見なします。

ClusterGroupUpgrade CR の **batchTimeoutAction** は、クラスターのアップグレードが失敗した場合にどうなるかを決定します。**continue** を指定して失敗したクラスターをスキップし、他のクラスターのアップグレードを続行するか、**abort** を指定してすべてのクラスターのポリシー修正を停止することができます。タイムアウトが経過すると、TALM はすべての強制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。

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-4.4.15.4
  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-4.15
          desiredUpdate:
            version: 4.4.15.4
          upstream: https://api.openshift.com/api/upgrades_info/v1/graph
        status:
          history:
            - state: Completed
              version: 4.4.15.4
          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

PolicyGenTemplate CRD の詳細は、[About the PolicyGenTemplate CRD](#) を参照してください。

11.6.1. 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。

11.6.2. マネージドクラスターへの更新ポリシーの適用

ポリシーを適用してマネージドクラスターを更新できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1つ以上のマネージドクラスターをプロビジョニングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. **ClusterGroupUpgrade** CR の内容を **cgu-1.yaml** ファイルに保存します。

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
  managedPolicies: 1
  - policy1-common-cluster-version-policy
  - policy2-common-nto-sub-policy
  - policy3-common-ntp-sub-policy
  - policy4-common-sriov-sub-policy

```

```

enable: false
clusters: ❷
- spoke1
- spoke2
- spoke5
- spoke6
remediationStrategy:
  maxConcurrency: 2 ❸
  timeout: 240 ❹
batchTimeoutAction: ❺

```

- ❶ 適用するポリシーの名前。
- ❷ 更新するクラスターのリスト。
- ❸ **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。
- ❹ 更新のタイムアウト (分単位)。
- ❺ 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", ❶
      "reason": "NotEnabled",
      "status": "False",
      "type": "Progressing"
    }
  ]
},

```

```

"copiedPolicies": [
  "cgu-policy1-common-cluster-version-policy",
  "cgu-policy2-common-nto-sub-policy",
  "cgu-policy3-common-ntp-sub-policy",
  "cgu-policy4-common-sriov-sub-policy"
],
"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-ntp-sub-policy": "[{"kind":"Subscription","name":"ntp-operator-subscription","namespace":"openshift-ntp"}]",
  "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-ntp-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-ntp-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-ntp-sub-policy",
  "cgu-policy4-common-sriov-sub-policy"
],
"placementRules": [
  "cgu-policy1-common-cluster-version-policy",
  "cgu-policy2-common-nto-sub-policy",
  "cgu-policy3-common-ntp-sub-policy",
  "cgu-policy4-common-sriov-sub-policy"
],
"precaching": {
  "spec": {}
},
"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。

- c. 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 | COMPLIANCE STATE | AGE | REMEDIATION ACTION |
|-----------|---|------------------|-----|---------------------|
| default | cgu-policy1-common-cluster-version-policy | 17m | ① | enforce |
| default | cgu-policy2-common-nto-sub-policy | 17m | | enforce |
| default | cgu-policy3-common-ntp-sub-policy | 17m | | enforce |
| default | cgu-policy4-common-sriov-sub-policy | 17m | | enforce |
| default | policy1-common-cluster-version-policy | 15h | | inform NonCompliant |
| default | policy2-common-nto-sub-policy | 15h | | inform NonCompliant |
| default | policy3-common-ntp-sub-policy | 18m | | inform NonCompliant |
| default | policy4-common-sriov-sub-policy | 18m | | inform NonCompliant |

- ① 現在クラスターに適用されるポリシーの **spec.remediationAction** フィールドは、**enforce** に設定されます。**ClusterGroupUpgrade** CR からの **inform** モードのマネージドポリシーは、更新中も **inform** モードで残ります。

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再度確認します。

```
$ 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"
    }
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ntp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "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-ntp-sub-policy": "[{"kind":"Subscription","name":"ntp-operator-subscription","namespace":"openshift-ntp"}]",
    "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-ntp-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"
],
"precaching": {
  "spec": {}
},
"remediationPlan": [
  [
    "spoke1",
    "spoke2"
  ],
  [
    "spoke5",
    "spoke6"
  ]
],
"status": {
  "currentBatch": 1,
  "currentBatchStartedAt": "2022-02-25T15:54:16Z",
  "remediationPlanForBatch": {
    "spoke1": 0,
    "spoke2": 1
  },
  "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. ポリシーに Operator サブスクリプションが含まれる場合、インストールの進捗を単一ノードクラスターで直接確認できます。
 - a. 以下のコマンドを実行して、インストールの進捗を確認する単一ノードクラスターの **KUBECONFIG** ファイルをエクスポートします。

```
$ export KUBECONFIG=<cluster_kubeconfig_absolute_path>
```

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。

- b. 単一ノードクラスターに存在するすべてのアノ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 | | | |

3. 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 | 4.4.15.5 | True | True | 43s | Working towards 4.4.15.7: 71 of 735 done (9% complete) |

4. 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}"
```

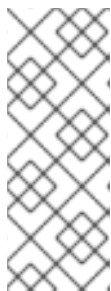
5. 以下のコマンドを実行して、必要なサブスクリプションに関連付けられている単一ノードのクラスターに存在するインストール計画を確認します。

```
$ oc get installplan -n <subscription_namespace>
```

cluster-logging Operator の出力例

| NAMESPACE | NAME | CSV | APPROVAL |
|-------------------|---------------|-------------------------|-------------|
| APPROVED | | | |
| openshift-logging | install-6khtw | cluster-logging.5.3.3-4 | Manual true |

- ① インストール計画の **Approval** フィールドは **Manual** に設定されており、TALM がインストール計画を承認すると、**Approved** フィールドは **false** から **true** に変わります。



注記

TALM がサブスクリプションを含むポリシーを修正している場合、そのサブスクリプションに関連付けられているすべてのインストールプランが自動的に承認されます。オペレーターが最新の既知のバージョンに到達するために複数のインストールプランが必要な場合、TALM は複数のインストールプランを承認し、最終バージョンに到達するために1つ以上の中間バージョンをアップグレードします。

6. 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 |

11.7. アップグレード前のクラスターリソースのバックアップの作成

単一ノードの OpenShift の場合、Topology Aware Lifecycle Manager (TALM) は、アップグレード前にデプロイメントのバックアップを作成できます。アップグレードが失敗した場合は、以前のバージョンを回復し、アプリケーションの再プロビジョニングを必要とせずにクラスターを動作状態に復元できます。

バックアップ機能を使用するには、最初に **backup** フィールドを **true** に設定して **ClusterGroupUpgrade** CR を作成します。バックアップの内容が最新であることを確認するために、**ClusterGroupUpgrade** CR の **enable** フィールドを **true** に設定するまで、バックアップは取得されません。

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

- **true**
 - すべてのクラスターのバックアップが完了したか、バックアップの実行が完了したが、1つ以上のクラスターで失敗しました。いずれかのクラスターでバックアップが失敗した場合、そのクラスターの更新は続行されません。
- **false**
 - 1つ以上のクラスターのバックアップがまだ進行中か、すべてのクラスターのバックアップが失敗しました。スポーククラスターで実行されているバックアッププロセスには、次のステータスがあります。
 - **PreparingToStart**
最初の調整パスが進行中です。TALM は、失敗したアップグレード試行で作成されたスポークバックアップネームスペースとハブビューリソースをすべて削除します。
 - **Starting**
バックアップの前提条件とバックアップジョブを作成しています。
 - **Active**
バックアップが進行中です。
 - **Succeeded**
バックアップは成功しました。
 - **BackupTimeout**
アーティファクトのバックアップは部分的に行われます。
 - **UnrecoverableError**
バックアップはゼロ以外の終了コードで終了しました。



注記

クラスタのバックアップが失敗し、**BackupTimeout** または **UnrecoverableError** 状態になると、そのクラスタのクラスタ更新は続行されません。他のクラスタへの更新は影響を受けず、続行されます。

11.7.1. バックアップを含む ClusterGroupUpgrade CR の作成

シングルノードの OpenShift クラスタでアップグレードする前に、デプロイメントのバックアップを作成できます。アップグレードが失敗した場合は、Topology Aware Lifecycle Manager (TALM) によって生成された **upgrade-recovery.sh** スクリプトを使用して、システムをアップグレード前の状態に戻すことができます。バックアップは次の項目で設定されます。

クラスタのバックアップ

etcd と静的 Pod マニフェストのスナップショット。

コンテンツのバックアップ

/etc、**/usr/local**、**/var/lib/kubelet** などのフォルダのバックアップ。

変更されたファイルのバックアップ

変更された **machine-config** によって管理されるすべてのファイル。

Deployment

固定された **ostree** デプロイメント。

イメージ (オプション)

使用中のコンテナイメージ。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1つ以上のマネージドクラスタをプロビジョニングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- Red Hat Advanced Cluster Management 2.2.4 をインストールします。



注記

リカバリーパーティションを作成することを強く推奨します。以下は、50 GB のリカバリーパーティションの **SiteConfig** カスタムリソース (CR) の例です。

```
nodes:
  - hostName: "node-1.example.com"
    role: "master"
    rootDeviceHints:
      hctl: "0:2:0:0"
      deviceName: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
  ...
  #Disk /dev/disk/by-id/scsi-3600508b400105e210000900000490000:
  #893.3 GiB, 959119884288 bytes, 1873281024 sectors
  diskPartition:
    - device: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
  partitions:
    - mount_point: /var/recovery
      size: 51200
      start: 800000
```

手順

1. **clustergroupupgrades-group-du.yaml** ファイルで、**backup** フィールドと **enable** フィールドを **true** に設定して、**ClusterGroupUpgrade** CR の内容を保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true
  backup: true
  clusters:
    - cnfdb1
    - cnfdb2
  enable: true
  managedPolicies:
    - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

2. 更新を開始するには、次のコマンドを実行して **ClusterGroupUpgrade** CR を適用します。

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

検証

- 以下のコマンドを実行して、ハブクラスターのアップグレードのステータスを確認します。

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

出力例

```
{
  "backup": {
    "clusters": [
      "cnfdb2",
      "cnfdb1"
    ],
    "status": {
      "cnfdb1": "Succeeded",
      "cnfdb2": "Failed" ❶
    }
  },
  "computedMaxConcurrency": 1,
  "conditions": [
    {
      "lastTransitionTime": "2022-04-05T10:37:19Z",
      "message": "Backup failed for 1 cluster", ❷
      "reason": "PartiallyDone", ❸
      "status": "True", ❹
      "type": "Succeeded"
    }
  ],
  "precaching": {
    "spec": {}
  },
  "status": {}
}
```

- ❶ 1つのクラスターのバックアップが失敗しました。
- ❷ このメッセージは、1つのクラスターのバックアップが失敗したことを確認します。
- ❸ バックアップは部分的に成功しました。
- ❹ バックアッププロセスが終了しました。

11.7.2. アップグレードが失敗した後のクラスターのリカバリー

クラスターのアップグレードが失敗した場合は、手動でクラスターにログインし、バックアップを使用してクラスターをアップグレード前の状態に戻すことができます。次の2つの段階があります。

ロールバック

試行されたアップグレードにプラットフォーム OS 展開への変更が含まれていた場合は、回復スクリプトを実行する前に、以前のバージョンにロールバックする必要があります。



重要

ロールバックは、TALM および単一ノード OpenShift からのアップグレードにのみ適用されます。このプロセスは、他のアップグレードタイプからのロールバックには適用されません。

復元

リカバリーはコンテナをシャットダウンし、バックアップパーティションのファイルを使用してコンテナを再起動し、クラスターを復元します。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1つ以上のマネージドクラスターをプロビジョニングします。
- Red Hat Advanced Cluster Management 2.2.4 をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- バックアップ用に設定されたアップグレードを実行します。

手順

1. 次のコマンドを実行して、以前に作成した **ClusterGroupUpgrade** カスタムリソース (CR) を削除します。

```
$ oc delete cgu/du-upgrade-4918 -n ztp-group-du-sno
```

2. リカバリーするクラスターにログインします。
3. 次のコマンドを実行して、プラットフォーム OS の展開のステータスを確認します。

```
$ ostree admin status
```

出力例

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9.0
  Version: 49.84.202202230006-0
  Pinned: yes ①
  origin refspeg:
c038a8f08458bbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9
```

- ① 現在の展開は固定されています。プラットフォーム OS 展開のロールバックは必要ありません。

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa.0
  Version: 410.84.202204050541-0
  origin refspeg: f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa
rhcos ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca.0
(rollback) ①
  Version: 410.84.202203290245-0
  Pinned: yes ②
  origin refspeg:
ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca
```

- ① このプラットフォーム OS の展開は、ロールバックの対象としてマークされています。

2 以前の展開は固定されており、ロールバックできます。

4. プラットフォーム OS 展開のロールバックをトリガーするには、次のコマンドを実行します。

```
$ rpm-ostree rollback -r
```

5. 復元の最初のフェーズでは、コンテナをシャットダウンし、ファイルをバックアップパーティションから対象のディレクトリに復元します。リカバリーを開始するには、次のコマンドを実行します。

```
$ /var/recovery/upgrade-recovery.sh
```

6. プロンプトが表示されたら、次のコマンドを実行してクラスターを再起動します。

```
$ systemctl reboot
```

7. 再起動後、次のコマンドを実行してリカバリーを再開します。

```
$ /var/recovery/upgrade-recovery.sh --resume
```



注記

リカバリーユーティリティが失敗した場合は、**--restart** オプションを使用して再試行できます。

```
$ /var/recovery/upgrade-recovery.sh --restart
```

検証

- リカバリーのステータスを確認するには、次のコマンドを実行します。

```
$ oc get clusterversion,nodes,clusteroperator
```

出力例

```
NAME                                VERSION AVAILABLE PROGRESSING SINCE
STATUS
clusterversion.config.openshift.io/version 4.4.15.23 True False 86d Cluster
version is 4.4.15.23 1
```

```
NAME                STATUS ROLES    AGE VERSION
node/lab-test-spoke1-node-0 Ready master,worker 86d v1.22.3+b93fd35 2
```

```
NAME                                VERSION AVAILABLE
PROGRESSING DEGRADED SINCE MESSAGE
clusteroperator.config.openshift.io/authentication 4.4.15.23 True
False False 2d7h 3
clusteroperator.config.openshift.io/baremetal 4.4.15.23 True False
False 86d
```



- ① クラスターのバージョンが利用可能であり、正しいバージョンを持っています。
- ② ノードのステータスは **Ready** です。
- ③ **ClusterOperator** オブジェクトの可用性は **True** です。

11.8. コンテナイメージ事前キャッシュ機能の使用

シングルノードの OpenShift クラスターでは、コンテナ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。

コンテナイメージの事前キャッシュは、**ClusterGroupUpgrade** CR で **preCaching** フィールドが **true** に設定されている場合に起動します。

TALM は **PrecacheSpecValid** 条件を使用して、次のようにステータス情報を報告します。

- **true**
事前キャッシュの仕様は有効で一貫性があります。
- **false**
事前キャッシュの仕様は不完全です。

TALM は **PrecachingSucceeded** 条件を使用して、次のようにステータス情報を報告します。

- **true**
TALM は事前キャッシュプロセスを完了しました。いずれかのクラスターで事前キャッシュが失敗した場合、そのクラスターの更新は失敗しますが、他のすべてのクラスターの更新は続行されます。クラスターの事前キャッシュが失敗した場合は、メッセージで通知されます。
- **false**
1つ以上のクラスターで事前キャッシュがまだ進行中か、すべてのクラスターで失敗しました。

事前キャッシュプロセスに成功すると、ポリシーの修正を開始できます。修復アクションは、**enable** フィールドが **true** に設定されている場合に開始されます。クラスターで事前キャッシュエラーが発生した場合、そのクラスターのアップグレードは失敗します。アップグレードプロセスは、事前キャッシュが成功した他のすべてのクラスターに対して続行されます。

事前キャッシュプロセスは、以下のステータスにあります。

- **NotStarted**
これは、すべてのクラスターが **ClusterGroupUpgrade** CR の最初の調整パスで自動的に割り当てられる初期状態です。この状態では、TALM は、以前の不完全な更新から残ったスポーククラスターの事前キャッシュの namespace およびハブビューリソースを削除します。次に TALM

は、スポーク前の namespace の新規の **ManagedClusterView** リソースを作成し、**PrecachePreparing** 状態の削除を確認します。

- **PreparingToStart**
以前の不完全な更新からの残りのリソースを消去すると進行中です。
- **Starting**
キャッシュ前のジョブの前提条件およびジョブが作成されます。
- **Active**
ジョブは Active の状態です。
- **Succeeded**
事前キャッシュジョブが成功しました。
- **PrecacheTimeout**
アーティファクトの事前キャッシュは部分的に行われます。
- **UnrecoverableError**
ジョブはゼロ以外の終了コードで終了します。

11.8.1. コンテナ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な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。

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

シングルノードの OpenShift の場合は、事前キャッシュ機能により、更新が開始する前に、必要なコンテナイメージをスポーククラスターに配置できます。



注記

事前キャッシュの場合、TALM は **ClusterGroupUpgrade** CR の **spec.remediationStrategy.timeout** 値を使用します。事前キャッシュジョブが完了するのに十分な時間を与える **timeout** 値を設定する必要があります。事前キャッシュの完了後に **ClusterGroupUpgrade** CR を有効にすると、**timeout** 値を更新に適した期間に変更できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1つ以上のマネージドクラスターをプロビジョニングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **clustergroupupgrades-group-du.yaml** ファイルで **preCaching** フィールドを **true** に設定して **ClusterGroupUpgrade** CR の内容を保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true ①
  clusters:
    - cnfdb1
    - cnfdb2
  enable: false
  managedPolicies:
    - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

- ① **preCaching** フィールドは **true** に設定されています。これにより、更新を開始する前に TALM がコンテナ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。

- 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": "PrecacheSpecsWellFormed",
      "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。

- 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" ❶
  }
]
```

- ❶ キャッシュ前のタスクが実行されます。

11.9. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング

Topology Aware Lifecycle Manager (TALM) は、RHACM ポリシーを修復する OpenShift Container Platform Operator です。問題が発生した場合には、**oc adm must-gather** コマンドを使用して詳細およびログを収集し、問題のデバッグ手順を行います。

関連トピックの詳細は、以下のドキュメントを参照してください。

- [Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix](#)
- [Red Hat Advanced Cluster Management Troubleshooting](#)
- Operator の問題のトラブルシューティングセクション

11.9.1. 一般的なトラブルシューティング

以下の質問を確認して、問題の原因を特定できます。

- 適用する設定がサポートされているか？
 - RHACM と OpenShift Container Platform のバージョンと互換性があるか？

- TALM および RHACM のバージョンと互換性があるか？
- 問題の原因となる以下のコンポーネントはどれですか？
 - 「管理ポリシー」
 - 「クラスター」
 - 「修復ストラテジー」
 - 「Topology Aware Lifecycle Manager」

ClusterGroupUpgrade 設定が機能するようにするには、以下を実行できます。

1. **spec.enable** フィールドを **false** に設定して **ClusterGroupUpgrade** CR を作成します。
2. ステータスが更新され、トラブルシューティングの質問を確認するのを待ちます。
3. すべてが予想通りに機能する場合は、**ClusterGroupUpgrade** CR で **spec.enable** フィールドを **true** に設定します。



警告

ClusterUpgradeGroup CR で **spec.enable** フィールドを **true** に設定すると、更新手順が起動し、CR の **spec** フィールドを編集することができなくなります。

11.9.2. ClusterUpgradeGroup CR を変更できません。

問題

更新を有効にした後に、**ClusterUpgradeGroup** CR を編集することはできません。

解決方法

以下の手順を実行して手順を再起動します。

1. 以下のコマンドを実行して古い **ClusterGroupUpgrade** CR を削除します。

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. マネージドクラスターおよびポリシーに関する既存の問題を確認し、修正します。
 - a. すべてのクラスターがマネージドクラスターで、利用可能であることを確認します。
 - b. すべてのポリシーが存在し、**spec.remediationAction** フィールドが **inform** に設定されていることを確認します。
3. 正しい設定で新規の **ClusterGroupUpgrade** CR を作成します。

```
$ oc apply -f <ClusterGroupUpgradeCR_YAML>
```


11.9.3. 管理ポリシー

システムでの管理ポリシーの確認

問題

システムで正しい管理ポリシーがあるかどうかをチェックする。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

出力例

```
["group-du-sno-validator-du-validator-policy", "policy2-common-nto-sub-policy", "policy3-common-
ptp-sub-policy"]
```

remediationAction モードの確認

問題

remediationAction フィールドが、管理ポリシーの **spec** で **inform** に設定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get policies --all-namespaces
```

出力例

| NAMESPACE | NAME | STATE | AGE | REMEDIATION ACTION | COMPLIANCE |
|-----------|---------------------------------------|-------|-------|--------------------|--------------------|
| default | policy1-common-cluster-version-policy | | 5d21h | inform | NonCompliant |
| 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 | STATE | AGE | REMEDIATION ACTION | COMPLIANCE |
|-----------|------|-------|-----|--------------------|------------|
|-----------|------|-------|-----|--------------------|------------|

| | | | | |
|------------------|---------------------------------------|--------|--------------|-------|
| default 5d21h | policy1-common-cluster-version-policy | inform | NonCompliant | |
| 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 |

11.9.4. クラスタ

マネージドクラスタが存在するかどうかの確認

問題

ClusterGroupUpgrade CR のクラスタがマネージドクラスタかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

| NAME | HUB ACCEPTED | MANAGED CLUSTER | URLS | JOINED | AVAILABLE |
|---------------|--------------|-------------------------------------|------|---------|-----------|
| AGE | | | | | |
| local-cluster | true | https://api.hub.example.com:6443 | True | Unknown | 13d |
| spoke1 | true | https://api.spoke1.example.com:6443 | True | True | 13d |
| spoke3 | true | https://api.spoke3.example.com:6443 | True | True | 27h |

1. または、TALM マネージャーログを確認します。

a. 以下のコマンドを実行して、TALM マネージャーの名前を取得します。

```
$ oc get pod -n openshift-operators
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp 2/2 Running 0
45m
```

b. 以下のコマンドを実行して、TALM マネージャーログを確認します。

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

出力例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error
{"reconciler group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade",
"name": "lab-upgrade", "namespace": "default", "error": "Cluster spoke5555 is not a
ManagedCluster"} 1
sigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).processNextWorkItem
```

- 1 エラーメッセージには、クラスターがマネージドクラスターではないことが分かります。

マネージドクラスターが利用可能かどうかの確認

問題

ClusterGroupUpgrade CR で指定されたマネージドクラスターが利用可能かどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

| NAME | HUB ACCEPTED | MANAGED CLUSTER URLS | JOINED | AVAILABLE |
|---------------|--------------|-------------------------------------|--------|-------------------|
| AGE | | | | |
| local-cluster | true | https://api.hub.testlab.com:6443 | True | Unknown 13d |
| spoke1 | true | https://api.spoke1.testlab.com:6443 | True | True 13d 1 |
| spoke3 | true | https://api.spoke3.testlab.com:6443 | True | True 27h 2 |

- 1
 - 2
- マネージドクラスターの **AVAILABLE** フィールドの値は **True** です。

clusterLabelSelector のチェック

問題

ClusterGroupUpgrade CR で指定された **clusterLabelSelector** フィールドが、管理対象クラスターの少なくとも1つと一致するかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get managedcluster --selector=upgrade=true 1
```

- 1
- 更新するクラスターのラベルは **upgrade:true** です。

出力例

| NAME | HUB ACCEPTED | MANAGED CLUSTER URLS | JOINED |
|---------------|--------------|-------------------------------------|---------------|
| AVAILABLE AGE | | | |
| spoke1 | true | https://api.spoke1.testlab.com:6443 | True True 13d |
| spoke3 | true | https://api.spoke3.testlab.com:6443 | True True 27h |

カナリアクラスターが存在するかどうかの確認

問題

カナリアクラスターがクラスターのリストに存在するかどうかを確認します。

ClusterGroupUpgrade CR の例

```
spec:
  remediationStrategy:
    canaries:
      - spoke3
    maxConcurrency: 2
    timeout: 240
  clusterLabelSelectors:
    - matchLabels:
        upgrade: true
```

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.clusters}'
```

出力例

```
["spoke1", "spoke3"]
```

1. 以下のコマンドを実行して、カナリアクラスターが **clusterLabelSelector** ラベルに一致するクラスターの一覧に存在するかどうかを確認します。

```
$ oc get managedcluster --selector=upgrade=true
```

出力例

| NAME | HUB ACCEPTED | MANAGED CLUSTER URLS | JOINED | AVAILABLE |
|--------|--------------|-------------------------------------|--------|-----------|
| spoke1 | true | https://api.spoke1.testlab.com:6443 | True | 13d |
| spoke3 | true | https://api.spoke3.testlab.com:6443 | 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があります。

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
```

11.9.5. 修復ストラテジー

remediationStrategy が ClusterGroupUpgrade CR に存在するかどうかの確認

問題

remediationStrategy が **ClusterGroupUpgrade** CR に存在するかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

出力例

```
{"maxConcurrency":2, "timeout":240}
```

ClusterGroupUpgrade CR に maxConcurrency が指定されているかどうかの確認

問題

maxConcurrency が **ClusterGroupUpgrade** CR で指定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy.maxConcurrency}'
```

出力例

```
2
```

11.9.6. Topology Aware Lifecycle Manager

ClusterGroupUpgrade CR での条件メッセージおよびステータスの確認

問題

ClusterGroupUpgrade CR の **status.conditions** フィールドの値を確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

出力例

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"Missing managed policies:[policyList]", "reason":"NotAllManagedPoliciesExist", "status":"False", "type":"Validated"}
```

対応するコピーされたポリシーの確認

問題

status.managedPoliciesForUpgrade からのすべてのポリシーに **status.copiedPolicies** に対応するポリシーがあ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 -oyaml
```

出力例

```
status:
  ...
  copiedPolicies:
  - lab-upgrade-policy3-common-ntp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy3-common-ntp-sub-policy
    namespace: default
```

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 -jsonpath='{.status.remediationPlan}'
```

出力例

```
[[{"spoke2", "spoke3"}]]
```

TALM マネージャーコンテナのエラー

問題

TALM のマネージャーコンテナのログを確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

出力例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error {"reconciler
group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade", "name": "lab-upgrade",
"namespace": "default", "error": "Cluster spoke5555 is not a ManagedCluster"} 1
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

1 エラーを表示します。

ClusterGroupUpgrade CR が完了した後、クラスターが一部のポリシーに準拠していない

====

問題

修復が必要かどうかを判断するために TALM が使用するポリシーコンプライアンスステータスは、まだすべてのクラスターで完全に更新されていません。これには次の理由が考えられます。

- ポリシーの作成または更新後、CGU の実行が早すぎました。
- ポリシーの修復は、**ClusterGroupUpgrade** CR の後続のポリシーのコンプライアンスに影響します。

解決方法

同じ仕様で新しい **ClusterGroupUpdate** CR を作成して適用します。

GitOps ZTP ワークフローで自動作成された ClusterGroupUpgrade CR に管理ポリシーがない

問題

クラスターが **Ready** になったときにマネージドクラスターのポリシーがない場合、ポリシーのない **ClusterGroupUpgrade** CR が自動作成されます。**ClusterGroupUpgrade** CR が完了すると、マネージドクラスターには **ztp-done** というラベルが付けられます。**SiteConfig** リソースがプッシュされた後、必要な時間内に **PolicyGenTemplate** CR が Git リポジトリにプッシュされなかった場合、クラスターが **Ready** になったときに、ターゲットクラスターで使用できるポリシーがなくなる可能性があります。

解決方法

適用するポリシーがハブクラスターで使用可能であることを確認してから、必要なポリシーを使用して **ClusterGroupUpgrade** CR を作成します。

ClusterGroupUpgrade CR を手動で作成するか、自動作成を再度トリガーすることができます。**ClusterGroupUpgrade** CR の自動作成をトリガーするには、クラスターから **ztp-done** ラベルを削除し、以前に **zip-install** 名前空間で作成された空の **ClusterGroupUpgrade** CR を削除します。

事前キャッシュに失敗しました

問題

事前キャッシュは、次のいずれかの理由で失敗する場合があります。

- ノードに十分な空き容量がありません。
- 切断された環境では、事前キャッシュイメージが適切にミラーリングされていません。
- Pod の作成中に問題が発生しました。

解決方法

1. スペース不足のために事前キャッシュが失敗したかどうかを確認するには、ノードの事前キャッシュ Pod のログを確認します。

- a. 次のコマンドを使用して Pod の名前を見つけます。

```
$ oc get pods -n openshift-talo-pre-cache
```

- b. 次のコマンドを使用してログをチェックし、エラーが容量不足に関連しているかどうかを確認します。

```
$ oc logs -n openshift-talo-pre-cache <pod name>
```

2. ログがない場合は、次のコマンドを使用して Pod のステータスを確認します。

```
$ oc describe pod -n openshift-talo-pre-cache <pod name>
```

3. Pod が存在しない場合は、次のコマンドを使用してジョブのステータスをチェックし、Pod を作成できなかった理由を確認します。

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

関連情報

- トラブルシューティングに関する詳細は、[Operator 関連の問題の OpenShift Container Platform トラブルシューティング](#) を参照してください。
- ZTP ワークフローで Topology Aware Lifecycle Manager を使用方法の詳細については、[Topology Aware Lifecycle Manager を使用した管理ポリシーの更新](#) を参照してください。
- **PolicyGenTemplate** CRD の詳細は、[About the PolicyGenTemplate CRD](#) を参照してください。

第12章 TOPOLOGY AWARE LIFECYCLE MANAGER を使用した非接続環境でのマネージドクラスターの更新

Topology Aware Lifecycle Manager (TALM) を使用して、OpenShift Container Platform 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) 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。

関連情報

- Topology Aware Lifecycle Manager の詳細は、[About the Topology Aware Lifecycle Manager](#) を参照してください。

12.1. 切断された環境でのクラスターの更新

GitOps Zero Touch Provisioning (ZTP) および Topology Aware Lifecycle Manager (TALM) を使用してデプロイしたマネージドクラスターとそのマネージドクラスターの Operator をアップグレードできます。

12.1.1. 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。
 - 必要な OpenShift Container Platform イメージリポジトリーをミラーリングします。追加リソースにリンクされている OpenShift Container Platform イメージリポジトリーのミラーリング手順に従って、目的のプラットフォームイメージがミラーリングされていることを確認してください。**imageContentSources.yaml** ファイルの **imageContentSources** セクションの内容を保存します。

出力例

```
imageContentSources:
  - mirrors:
    - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
    source: quay.io/openshift-release-dev/ocp-release
  - mirrors:
    - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

- 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。
 - 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>
```

- 次のコマンドを実行して、クラスターのアーキテクチャーを指定します。

■

```
$ ARCHITECTURE=<cluster_architecture> 1
```

- 1 **x86_64**、**aarch64**、**s390x**、または **ppc64le** など、クラスターのアーキテクチャーを指定します。

- c. 次のコマンドを実行して、Quay からリリースイメージダイジェストを取得します。

```
$ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From: .*@/p')"
```

- d. 次のコマンドを実行して、ダイジェストアルゴリズムを設定します。

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

- e. 次のコマンドを実行して、ダイジェスト署名を設定します。

```
$ DIGEST_ENCODED="${DIGEST#*:*}"
```

- f. 次のコマンドを実行して、mirror.openshift.com Web サイトからイメージ署名を取得します。

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

- g. 以下のコマンドを実行して、イメージ署名を **checksum-
<OCP_RELEASE_NUMBER>.yaml** ファイルに保存します。

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 更新グラフを準備します。更新グラフを準備するオプションは 2 つあります。

- a. OpenShift Update Service を使用します。
ハブクラスターでグラフを設定する方法の詳細については、[OpenShift Update Service の Operator のデプロイ](#) および [グラフデータ init コンテナのビルド](#) 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-4.15 -o ~/upgrade-graph_stable-4.15
```

- Operator の更新については、以下のタスクを実行する必要があります。
 - Operator カタログをミラーリングします。切断されたクラスターで使用する Operator カタログのミラーリングセクションの手順に従って、目的の Operator イメージがミラーリングされていることを確認します。

関連情報

- GitOps Zero Touch Provisioning (ZTP) の更新方法について、詳しくは [GitOps ZTP のアップグレード](#) を参照してください。
- OpenShift Container Platform イメージリポジトリ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。
- 更新チャンネルとリリースの詳細は、[更新チャンネルとリリースについて](#) を参照してください。

12.1.2. プラットフォームの更新の実行

TALM を使用してプラットフォームの更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して1つ以上のマネージドクラスターをプロビジョニングします。
- 目的のイメージリポジトリをミラーリングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新用の **PolicyGenTemplate** CR を作成します。
 - a. 次の **PolicyGenTemplate** CR の内容を **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} ❷
- fileName: DisconnectedICSP.yaml
  policyName: "platform-upgrade-prep"
  metadata:
    name: disconnected-internal-icsp-for-ocp
  spec:
    repositoryDigestMirrors: ❸
    - 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 ❹
  policyName: "platform-upgrade"
  metadata:
    name: version
  spec:
    channel: "stable-4.15"
    upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.15
    desiredUpdate:
      version: 4.15.4
  status:
    history:
      - version: 4.15.4
        state: "Completed"

```

- ❶ **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。
- ❷ 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。
- ❸ 目的の OpenShift Container Platform イメージを含むミラーリポジトリを表示します。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。
- ❹ 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のイメージコンテンツソースを作成し、目的の更新チャンネルと切断された環境でマネージドクラスターが到達可能な更新グラフを使用してクラスターバージョンを更新します。
 - **du-upgrade-platform-upgrade** ポリシーは、プラットフォームのアップグレードを実行するために使用されます。
- b. **PolicyGenTemplate** CR の GitOps ZTP Git リポジトリにある **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 リポジトリにプッシュします。

ArgoCD は Git リポジトリから変更を取得し、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。

12.1.3. Operator 更新の実行

TALM で Operator の更新を実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して1つ以上のマネージドクラスターをプロビジョニングします。
- 目的のインデックスイメージ、バンドルイメージ、およびバンドルイメージで参照されるすべての Operator イメージをミラーリングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. Operator の更新用に **PolicyGenTemplate** CR を更新します。
 - a. **du-upgrade.yaml** ファイルの次の追加コンテンツで **du-upgradePolicyGenTemplate** CR を更新します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
    mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v4.15 1
```

```

updateStrategy: ❷
  registryPoll:
    interval: 1h
status:
  connectionState:
    lastObservedState: READY ❸

```

- ❶ 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。
 - ❷ 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。
 - ❸ カタログ接続が最後に監視された状態。**READY** 値は、**CatalogSource** ポリシーの準備が整っていることを保証し、インデックス Pod がプルされ、実行中であることを示します。このように、TALM は最新のポリシー準拠状態に基づいて Operator をアップグレードします。
- b. この更新により、1つのポリシー **du-upgrade-operator-catsrc-policy** が生成され、必要な Operator イメージを含む新しいインデックスイメージで **redhat-operators** カタログソースが更新されます。



注記

Operator にイメージの事前キャッシュを使用する必要があり、**redhat-operators** 以外の別のカタログソースからの Operator がある場合は、次のタスクを実行する必要があります。

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

たとえば、目的の SRIOV-FEC Operator は、**certified-operators** カタログソースで入手できます。カタログソースと Operator サブスクリプションを更新するには、次の内容を追加して、2つのポリシー **du-upgrade-fec-catsrc-policy** と **du-upgrade-subscriptions-fec-policy** を生成します。

```

apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""

```

```

mcp: "master"
remediationAction: inform
sourceFiles:
  ...
  - fileName: DefaultCatsrc.yaml
    remediationAction: inform
    policyName: "fec-catsrc-policy"
    metadata:
      name: certified-operators
    spec:
      displayName: Intel SRIOV-FEC Operator
      image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
      updateStrategy:
        registryPoll:
          interval: 10m
  - fileName: AcceleratorsSubscription.yaml
    policyName: "subscriptions-fec-policy"
    spec:
      channel: "stable"
      source: certified-operators

```

- c. 共通の **PolicyGenTemplate** CR に指定されたサブスクリプションチャン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ンネルが更新に使用されます。



注記

GitOps ZTP 4.15 で適用される Operator のデフォルトチャンネルは、**performance-addon-operator** を除きすべて **stable** です。OpenShift Container Platform 4.11 以降、**performance-addon-operator** 機能は **node-tuning-operator** に移動されました。4.10 リリースの場合、PAO のデフォルトチャンネルは **v4.10** です。共通の **PolicyGenTemplate** CR でデフォルトのチャンネルを指定することもできます。

- d. **PolicyGenTemplate** CR の更新を GitOps ZTP Git リポジトリにプッシュします。ArgoCD は Git リポジトリから変更を取得し、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 ❶
  - common-subscriptions-policy ❷
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false

```

- ❶ 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。

- ❷ 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。
- [ポリシーのコンプライアンス状態が古いために Operator が更新されない場合のトラブルシューティング](#)。

12.1.3.1. ポリシーのコンプライアンス状態が古いために Operator が更新されない場合のトラブルシューティング

一部のシナリオでは、ポリシーのコンプライアンス状態が古いため、Topology Aware Lifecycle Manager (TALM) が Operator の更新を見逃す可能性があります。

カタログソースの更新後に Operator Lifecycle Manager (OLM) がサブスクリプションステータスを更新すると、時間がかかります。TALM が修復が必要かどうかを判断する間、サブスクリプションポリシーのステータスは準拠していると表示される場合があります。その結果、サブスクリプションポリシーで指定された Operator はアップグレードされません。

このシナリオを回避するには、別のカタログソース設定を **PolicyGenTemplate** に追加し、更新が必要な Operator のサブスクリプションでこの設定を指定します。

手順

1. **PolicyGenTemplate** リソースにカタログソース設定を追加します。

```
- fileName: DefaultCatsrc.yaml
  remediationAction: inform
  policyName: "operator-catsrc-policy"
  metadata:
    name: redhat-operators
  spec:
```

```

displayName: Red Hat Operators Catalog
image: registry.example.com:5000/olm/redhat-operators:v{product-version}
updateStrategy:
  registryPoll:
    interval: 1h
status:
  connectionState:
    lastObservedState: READY
- fileName: DefaultCatsrc.yaml
  remediationAction: inform
  policyName: "operator-catsrc-policy"
  metadata:
    name: redhat-operators-v2 ❶
  spec:
    displayName: Red Hat Operators Catalog v2 ❷
    image: registry.example.com:5000/olredhat-operators:<version> ❸
    updateStrategy:
      registryPoll:
        interval: 1h
    status:
      connectionState:
        lastObservedState: READY

```

- ❶ 新しい設定の名前を更新します。
- ❷ 新しい設定の表示名を更新します。
- ❸ 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-namespace
# ...
spec:
  source: redhat-operators-v2 ❶
# ...

```

- ❶ **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。

12.1.4. プラットフォームと Operator の更新を一緒に実行する

プラットフォームと Operator の更新を同時に実行できます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。

- GitOps Zero Touch Provisioning (ZTP) を最新バージョンに更新します。
- GitOps ZTP を使用して1つ以上のマネージドクラスターをプロビジョニングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新の実行および Operator 更新の実行セクションで説明されている手順に従って、更新用の **PolicyGenTemplate** CR を作成します。
2. プラットフォームの準備作業と Operator の更新を適用します。
 - a. プラットフォームの更新の準備作業、カタログソースの更新、およびターゲット クラスターのポリシーを含む **ClusterGroupUpgrade** CR の内容を **cgu-platform-operator-upgrade-prep.yml** ファイルに保存します。次に例を示します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade-prep
    - du-upgrade-operator-catsrc-policy
  clusterSelector:
    - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

- b. 次のコマンドを実行して、**cgu-platform-operator-upgrade-prep.yml** ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

- c. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

3. プラットフォーム用の **ClusterGroupUpdate** CR と、**spec.enable** フィールドを **false** に設定した Operator 更新を作成します。
 - a. 次の例に示すように、ポリシーとターゲットクラスターを含むプラットフォームと Operator の更新 **ClusterGroupUpdate** CR の内容を **cgu-platform-operator-upgrade.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
```

```

namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade ❶
    - du-upgrade-operator-catsrc-policy ❷
    - common-subscriptions-policy ❸
  preCaching: true
  clusterSelector:
    - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false

```

- ❶ これはプラットフォーム更新ポリシーです。
- ❷ 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。
- ❸ 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。

12.1.5. デプロイされたクラスターから Performance Addon Operator サブスクリプションを削除する

以前のバージョンの OpenShift Container Platform では、Performance Addon Operator はアプリケーションの自動低レイテンシーパフォーマンスチューニングを提供していました。OpenShift Container Platform 4.11 以降では、これらの機能は Node Tuning Operator の一部です。

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



注記

Operator の再インストールを防ぐために、Performance Addon Operator サブスクリプションを作成するポリシーを削除する必要があります。

参照 DU プロファイルには、**PolicyGenTemplate** CR **common-ranGen.yaml** に Performance Addon Operator が含まれています。デプロイされたマネージドクラスターからサブスクリプションを削除するには、**common-ranGen.yaml** を更新する必要があります。



注記

Performance Addon Operator 4.10.3-5 以降を OpenShift Container Platform 4.11 以降にインストールする場合、Performance Addon Operator はクラスターのバージョンを検出し、Node Tuning Operator 機能との干渉を避けるために自動的に休止状態になります。ただし、最高のパフォーマンスを確保するには、OpenShift Container Platform 4.11 クラスターから Performance Addon Operator を削除してください。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリを作成している。リポジトリ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にマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。 **common-subscriptions-policy** ポリシーのステータスが **Non-Compliant** に変わります。
3. Topology Aware Lifecycle Manager を使用して、ターゲットクラスターに変更を適用します。設定変更のロールアウトの詳細については、「関連情報」セクションを参照してください。
4. プロセスを監視します。ターゲットクラスターの **common-subscriptions-policy** ポリシーのステータスが **Compliant** の場合、Performance Addon Operator はクラスターから削除されています。次のコマンドを実行して、 **common-subscriptions-policy** のステータスを取得します。

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

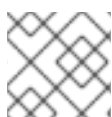
5. **common-ranGen.yaml** ファイルの **.spec.sourceFiles** から Performance Addon Operator namespace、Operator グループ、およびサブスクリプション CR を削除します。
6. 変更をカスタムサイトリポジトリにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。ポリシーは準拠したままです。

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

アプリケーションをアップグレードする前に、アプリケーション固有のワークロードイメージを単一ノード OpenShift クラスターに事前キャッシュできます。

次のカスタムリソース (CR) を使用して、事前キャッシュジョブの設定オプションを指定できます。

- **PreCachingConfig** CR
- **ClusterGroupUpgrade** CR



注記

PreCachingConfig CR のフィールドはすべてオプションです。

PreCachingConfig CR の例

```
apiVersion: ran.openshift.io/v1alpha1
```



```

kind: PreCachingConfig
metadata:
  name: exampleconfig
  namespace: exampleconfig-ns
spec:
  overrides: ❶
    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 ❷
  excludePrecachePatterns: ❸
    - aws
    - vsphere
  additionalImages: ❹
    -
quay.io/exampleconfig/application1@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925
ada29c559a47e2e1ef
    -
quay.io/exampleconfig/application2@sha256:3d5800123dee7cd4727d3fe238a97e2d2976d3808fc925
ada29c559a47adfaef
    -
quay.io/exampleconfig/applicationN@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdedga099
1234afdadsa09

```

- ❶ 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。
- ❷ 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。
- ❸ 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。
- ❹ 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 ❶
  preCachingConfigRef:
    name: exampleconfig ❷
    namespace: exampleconfig-ns ❸

```

- 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。

12.1.6.1. 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@sha256:4fe1334adfafadsf987123adfffdaf1243340adfafdga0991234afdadfsa09

```

- 1 **namespace** は、ハブクラスターにアクセスできる必要があります。
 - 2 事前にキャッシュされたイメージ用に十分なストレージ領域を確保するために、必要な最小ディスク領域フィールドを設定することを推奨します。
2. **preCaching** フィールドを **true** に設定して **ClusterGroupUpgrade** CR を作成し、前の手順で作成した **PreCachingConfig** CR を指定します。

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu
  namespace: default
spec:
  clusters:
  - sno1
  - sno2
  preCaching: true
  preCachingConfigRef:
  - name: exampleconfig
    namespace: default
  managedPolicies:

```

```
- du-upgrade-platform-upgrade
- du-upgrade-operator-catsrc-policy
- common-subscriptions-policy
remediationStrategy:
  timeout: 240
```



警告

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

3. イメージを事前キャッシュを開始する場合は、次のコマンドを実行して **ClusterGroupUpgrade** CR CR を適用します。

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

TALM は **ClusterGroupUpgrade** CR を検証します。

この時点から、TALM 事前キャッシュワークフローを続行できます。



注記

すべてのサイトが同時に事前キャッシュされます。

検証

1. 次のコマンドを実行して、**ClusterUpgradeGroup** CR が適用されているハブクラスターの事前キャッシュステータスを確認します。

```
$ oc get cgu <cgu_name> -n <cgu_namespace> -oyaml
```

出力例

```
precaching:
  spec:
    platformImage: quay.io/openshift-release-dev/ocp-
release@sha256:3d5800990dee7cd4727d3fe238a97e2d2976d3808fc925ada29c559a47e2e1
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: PrecacheSpecsWellFormed
  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"

```

- 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 |

- 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 | SuccessfulCreate | 19s | job-controller | Created pod: pre-cache-abcd1 |

- 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
```

- 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 | SuccessfulCreate | 5m19s | job-controller | Created pod: pre-cache-abcd1 |
| Normal | Completed | 19s | job-controller | Job completed |

- イメージが単一ノード OpenShift で正常に事前キャッシュされていることを確認するには、次の手順を実行します。

- デバッグモードでノードに入ります。

```
$ oc debug node/cnfd00.example.lab
```

- root を **host** に変更します。

```
$ chroot /host/
```

- 目的のイメージを検索します。

```
$ sudo podman images | grep <operator_name>
```

関連情報

- TALM の事前キャッシュワークフローについて、詳細は [コンテナ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。

12.2. GITOPS ZTP 用に自動作成された CLUSTERGROUPUPGRADE CR について

TALM には、**ManagedClusterForCGU** と呼ばれるコントローラーがあります。このコントローラーは、ハブクラスター上で **ManagedCluster** CR の **Ready** 状態を監視し、GitOps Zero Touch Provisioning (ZTP) の **ClusterGroupUpgrade** CR を作成します。

ztp-done ラベルが適用されていない **Ready** 状態のマネージドクラスターの場合、**ManagedClusterForCGU** コントローラーは、**ztp-install** namespace に **ClusterGroupUpgrade** CR と、GitOps ZTP プロセス中に作成された関連する RHACM ポリシーを自動的に作成します。次に TALM は自動作成された **ClusterGroupUpgrade** CR に一覧表示されている設定ポリシーのセットを修正し、設定 CR をマネージドクラスターにプッシュします。

クラスターが **Ready** になった時点でマネージドクラスターのポリシーがない場合、ポリシーのない **ClusterGroupUpgrade** CR が作成されます。**ClusterGroupUpgrade** が完了すると、マネージドクラスターには **ztp-done** というラベルが付けられます。そのマネージドクラスターに適用するポリシーがある場合は、2 日目の操作として **ClusterGroupUpgrade** を手動で作成します。

GitOps ZTP 用に自動作成された ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" ①
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" ②
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

- 1 TALM がクラスター設定を完了する際にマネージドクラスターに適用されます。
- 2 TALM が設定ポリシーのデプロイを開始するときにマネージドクラスターに適用されます。

第13章 GITOPS ZTP を使用した単一ノードの OPENSIFT クラスターの拡張

GitOps Zero Touch Provisioning (ZTP) を使用して、シングルノード OpenShift クラスターを拡張できます。単一ノードの OpenShift クラスターにワーカーノードを追加すると、元の単一ノードの OpenShift クラスターがコントロールプレーンノードのロールを保持します。ワーカーノードを追加しても、既存の単一ノード OpenShift クラスターのダウンタイムは必要ありません。



注記

単一ノードの OpenShift クラスターに追加できるワーカーノードの数に指定された制限はありませんが、追加のワーカーノード用にコントロールプレーンノードで予約されている CPU 割り当てを再評価する必要があります。

ワーカーノードでワークロードパーティショニングが必要な場合は、ノードをインストールする前に、ハブクラスターでマネージドクラスターポリシーをデプロイして修正する必要があります。そうすることで、GitOps ZTP ワークフローが **MachineConfig** ignition ファイルをワーカーノードに適用する前に、ワークロードパーティショニング **MachineConfig** オブジェクトがレンダリングされ、**worker** マシン設定プールに関連付けられます。

最初にポリシーを修正してから、ワーカーノードをインストールすることを推奨します。ワーカーノードのインストール後にワークロードパーティショニングマニフェストを作成する場合は、ノードを手動でドレインし、デーモンセットによって管理されるすべての Pod を削除する必要があります。管理デーモンセットが新しい Pod を作成すると、新しい Pod はワークロードパーティショニングプロセスを実行します。



重要

GitOps ZTP を使用した単一ノードの OpenShift クラスターへのワーカーノードの追加は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

関連情報

- vDU アプリケーションのデプロイ用に調整された単一ノードの OpenShift クラスターの詳細は、[単一ノードの OpenShift に vDU をデプロイするためのリファレンス設定](#) を参照してください。
- ワーカーノードの詳細は、[シングルノード OpenShift クラスターへのワーカーノードの追加](#) を参照してください。
- 拡張された単一ノードの 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たマネージドクラスターノードの削除](#) を参照してください。

13.1. ワーカーノードへのプロファイルの適用

DU プロファイルを使用して、追加のワーカーノードを設定できます。

GitOps Zero Touch Provisioning (ZTP) 共通、グループ、およびサイト固有の **PolicyGenTemplate** リソースを使用して、RAN 分散ユニット (DU) プロファイルをワーカーノードクラスターに適用できます。ArgoCD **policies** アプリケーションにリンクされている GitOps ZTP パイプラインには、**ztp-site-generate** コンテナを抽出するときに **out/argocd/example/policygentemplates** フォルダにある次の CR が含まれています。

- **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PLOY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クター設定を確認します。

```
$ oc get ptppoperatorconfig/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PLOY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,"enableOperatorWebhook":true} ❶
```

- ❶ ノードセクターが **master** に設定されている場合、スポークは、変更が必要なバージョンの GitOps ZTP プラグインでデプロイされています。

3. グループポリシーで、次の **ComplianceType** および **spec** エントリーを追加します。

```
spec:
- fileName: PtpOperatorConfig.yaml
  policyName: "config-policy"
  complianceType: mustonlyhave
  spec:
    daemonNodeSelector:
      node-role.kubernetes.io/worker: ""
- fileName: SrioVOperatorConfig.yaml
  policyName: "config-policy"
  complianceType: mustonlyhave
  spec:
    configDaemonNodeSelector:
      node-role.kubernetes.io/worker: ""
```



重要

daemonNodeSelector フィールドを変更すると、一時的な PTP Synchronization が失われ、SR-IOV 接続が失われます。

4. Git で変更をコミットし、GitOps ZTP ArgoCD アプリケーションによって監視されている Git リポジトリにプッシュします。

13.3. PTP および SR-IOV ノードセクターの互換性

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. 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

ワーカーノードのポリシーを作成できます。

手順

1. 次のポリシーテンプレートを作成します。

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
```



```

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-ntp=0:f:10*:ice-ntp.*
      [service]
      service.stalld=start,enable
      service.chrond=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 リポジトリにプッシュします。
5. 新しいポリシーをスポーククラスターに修正するには、TALM カスタムリソースを作成します。

```

$ cat <<EOF | oc apply -f -
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: example-sno-worker-policies
  namespace: default
spec:
  backup: false
  clusters:
  - example-sno
  enable: true

```

```

managedPolicies:
- group-du-sno-config-policy
- example-sno-workers-config-policy
- example-sno-config-policy
preCaching: false
remediationStrategy:
  maxConcurrency: 1
EOF

```

13.5. GITOPS ZTP を使用して単一ノードの OPENSIFT クラスターにワーカーノードを追加する

1つ以上のワーカーノードを既存の単一ノード OpenShift クラスターに追加して、クラスターで使用可能な CPU リソースを増やすことができます。

前提条件

- OpenShift Container Platform 4.11 以降のベアメタルハブクラスターに RHACM 2.6 以降をインストールして設定する
- ハブクラスターに Topology Aware Lifecycle Manager をインストールする
- ハブクラスターに Red Hat OpenShift GitOps をインストールする
- GitOps ZTP **ztp-site-generate** コンテナイメージバージョン 4.12 以降を使用する
- GitOps ZTP を使用して管理対象の単一ノード OpenShift クラスターをデプロイする
- RHACM ドキュメントの説明に従って、中央インフラストラクチャー管理を設定する
- 内部 API エンドポイント **api-int.<cluster_name>.<base_domain>** を解決するようにクラスターにサービスを提供する DNS を設定する

手順

1. **example-sno.yaml SiteConfig** マニフェストを使用してクラスターをデプロイした場合は、新しいワーカーノードを **spec.clusters[example-sno].nodes** リストに追加します。

```

nodes:
- hostName: "example-node2.example.com"
  role: "worker"
  bmcAddress: "idrac-virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.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      true    69m
example-node2 provisioning    true      true    4m50s 1
```

- 1** **provisioning** ステータスは、インストールメディアからのノードの起動が進行中であることを示します。

- インストールプロセスを継続的に監視します。
 - a. 次のコマンドを実行して、エージェントのインストールプロセスを監視します。

```
$ oc get agent -n example-sno --watch
```

出力例

```
NAME          CLUSTER APPROVED  ROLE  STAGE
671bc05d-5358-8940-ec12-d9ad22804faa example-sno true   master Done
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Starting
installation
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Installing
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Writing image
to disk
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Waiting for
control plane
[...]
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Rebooting
14fd821b-a35d-9cba-7978-00ddf535ff37 example-sno true   worker Done
```

- b. ワーカーノードのインストールが完了すると、ワーカーノードの証明書が自動的に承認されます。この時点で、ワーカーは **ManagedClusterInfo** ステータスで表示されます。次のコマンドを実行して、ステータスを確認します。

```
$ oc get managedclusterinfo/example-sno -n example-sno -o \
jsonpath='{range .status.nodeList[*]}.{.name}{"\t"}{.conditions}{"\t"}{.labels}{"\n"}{end}'
```

出力例

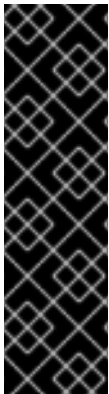
```
example-sno [{"status":"True","type":"Ready"}] {"node-
role.kubernetes.io/master":"","node-role.kubernetes.io/worker":""}
example-node2 [{"status":"True","type":"Ready"}] {"node-role.kubernetes.io/worker":""}
```


第14章 単一ノードの OPENSIFT デプロイメント用のイメージの事前キャッシュ

GitOps Zero Touch Provisioning (ZTP) ソリューションを使用して多数のクラスターをデプロイする、帯域幅が制限された環境では、OpenShift Container Platform のブートストラップとインストールに必要なすべてのイメージをダウンロードすることを避ける必要があります。リモートの単一ノードの OpenShift サイトでは帯域幅が制限されているため、デプロイに時間がかかる場合があります。factory-precaching-cli ツールを使用すると、ZTP プロビジョニングのためにサーバーをリモートサイトに出荷する前にサーバーを事前にステージングできます。

factory-precaching-cli ツールは次のことを行います。

- 最小限の ISO の起動に必要な RHCOS rootfs イメージをダウンロードします。
- **data** というラベルの付いたインストールディスクからパーティションを作成します。
- ディスクを xfs でフォーマットします。
- ディスクの最後に GUID パーティションテーブル (GPT) データパーティションを作成します。パーティションのサイズはツールで設定できます。
- OpenShift Container Platform のインストールに必要なコンテナ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イメージをコピーします。
- オプション: Day-2 Operator をパーティションにコピーします。



重要

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

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

14.1. FACTORY-PRE-CACHING-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。コンテナイメージ内の factory-precaching-cli ツール Go バイナリーは、**podman** を使用して RHCOS ライブイメージを実行しているサーバー上で実行されます。切断された環境で作業している場合、またはプライベートレジストリーがある場合は、そこにイメージをコピーして、イメージをサーバーにダウンロードできるようにする必要があります。

手順

- 次のコマンドを実行して、factory-precaching-cli ツールイメージをプルします。

```
# podman pull quay.io/openshift-kni/telco-ran-tools:latest
```

検証

- ツールが利用可能であることを確認するには、factory-precaching-cli ツール Go バイナリーの現在のバージョンを照会します。

```
# podman run quay.io/openshift-kni/telco-ran-tools:latest -- factory-precaching-cli -v
```

出力例

```
factory-precaching-cli version 20221018.120852+main.feecf17
```

14.2. ライブオペレーティングシステムイメージからの起動

factory-precaching-cli ツールを使用して、1つのディスクしか使用できず、外部ディスクドライブをサーバーに接続できないサーバーを起動できます。



警告

RHCOS では、ディスクが RHCOS イメージで書き込まれようとしているときに、ディスクが使用されていない必要があります。

サーバーハードウェアに応じて、次のいずれかの方法を使用して、空のサーバーに RHCOS ライブ ISO をマウントできます。

- Dell サーバーで Dell RACADM ツールを使用する。
- HP サーバーで HPONCFG ツールを使用する。
- Redfish BMC API を使用する。



注記

マウント手順を自動化することを推奨します。手順を自動化するには、必要なイメージをプルして、ローカル HTTP サーバーでホストする必要があります。

前提条件

- ホストの電源を入れた。
- ホストへのネットワーク接続がある。



手順

この例の手順では、Redfish BMC API を使用して RHCOS ライブ ISO をマウントします。

1. RHCOS ライブ ISO をマウントします。

、仮想メディアのフッターを確認！

- d. 仮想メディアのイメージをダウンロードして確認します。

```
$ 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 の作成の詳細は、[リモートサーバーアクセス用のカスタムライブ RHCOS ISO の作成](#) を参照してください。
- Dell RACADM ツールの使用の詳細については、[Integrated Dell Remote Access Controller 9 RACADM CLI Guide](#) を参照してください。
- HP HPONCFG ツールの使用の詳細については、[HPONCFG の使用](#) を参照してください。
- Redfish BMC API の使用に関する詳細は、[Redfish API を使用した HTTP ホスト ISO イメージからの起動](#) を参照してください。

14.3. ディスクのパーティション設定

完全な事前キャッシュプロセスを実行するには、ライブ ISO から起動し、コンテナイメージから `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

- パーティション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、コンテナは **privileged** として実行する必要があります。
- コンテナ内でプロセスを実行できるように、**/dev** フォルダ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 \ ❶
-d /dev/nvme0n1 \ ❷
-s 250 ❸
```

- ❶ factory-precaching-cli ツールのパーティショニング機能を指定します。
- ❷ ディスク上のルートディレクトリーを定義します。
- ❸ 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 (MCE) のバージョンを知る必要があります。`--du-profile` フラグを使用する場合は、単一ノードの OpenShift をプロビジョニングするハブクラスターで実行されている Red Hat Advanced Cluster Management (RHACM) のバージョンも指定する必要があります。

前提条件

- RHACM と MCE 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ました。
- 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ります。

- ベアメタルサーバーをインターネットに接続した。
- 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 と MCE の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 \ ❶
  -r 4.15.0 \ ❷
  --acm-version 2.6.3 \ ❸
  --mce-version 2.1.4 \ ❹
  -f /mnt \ ❺
  --img quay.io/custom/repository ❻
```

- ❶ factory-precaching-cli ツールのダウンロード機能を指定します。
- ❷ OpenShift Container Platform リリースバージョンを定義します。
- ❸ RHACM バージョンを定義します。
- ❹ MCE バージョンを定義します。
- ❺ 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。
- ❻ オプション: 追加のイメージを保存するリポジトリを定義します。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: | 4.15.0 |
| 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_f44dcf2c94e4fd843cbbf9b11128df2ba856cd813786e42e3da1fdb0f6ddd01.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.tg
z
-rw-r--r--. 1 root root 157937478 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_f7ba33a6a9db9cfc4b0ab0f368569e19b9fa08f4c01a0d5f6a243d61ab781bd8.tgz
```

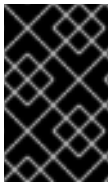
```
-rw-r--r--. 1 root root 145535253 Oct 31 15:26 ocp-v4.0-art-
dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de5.t
gz
-rw-r--r--. 1 root root 158048761 Oct 31 15:40 ocp-v4.0-art-
dev@sha256_f914228ddb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb47.tgz

-rw-r--r--. 1 root root 167914526 Oct 31 15:37 ocp-v4.0-art-
dev@sha256_fa3ca9401c7a9efda0502240aeb8d3ae2d239d38890454f17fe5158b62305010.tg
z
-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_b86b8aea29a818a9c22944fd18243fa0347c7a2bf1ad8864113ff2bb2d8
e0726.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 MCE Operator の適切なコンテナ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 MCE Operator 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 \ ❶
-r 4.15.0 \ ❷
--acm-version 2.6.3 \ ❸
--mce-version 2.1.4 \ ❹
-f /mnt \ ❺
--img quay.io/custom/repository ❻
--du-profile -s ❼
```

- ❶ factory-precaching-cli ツールのダウンロード機能を指定します。
- ❷ OpenShift Container Platform リリースバージョンを定義します。
- ❸ RHACM バージョンを定義します。
- ❹ MCE バージョンを定義します。
- ❺ 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。
- ❻ オプション: 追加のイメージを保存するリポジトリを定義します。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_0c81c2b79f79307305e51ce9d3837657cf9ba5866194e464b4d1b299f85034
d0
Processing artifact [379/379]: multicluster-operators-channel-
rhel8@sha256_c10f6bbb84fe36e05816e873a72188018856ad6aac6cc16271a1b3966f73ceb3

...
Summary:

Release:                4.15.0
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 4.15.0 \ 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 MCE 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-4.15
        minVersion: 4.15.0 1
        maxVersion: 4.15.0
  additionalImages:
    - name: quay.io/custom/repository
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.15
      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-4.15'
- 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:v4.15
  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 および MCE 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:v4.15
      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 \ ❶
-r 4.15.0 \ ❷
--acm-version 2.6.3 \ ❸
--mce-version 2.1.4 \ ❹
-f /mnt \ ❺
--img quay.io/custom/repository ❻
--du-profile -s \ ❼
--skip-imageset ❽
```

- ❶ factory-precaching-cli ツールのダウンロード機能を指定します。
- ❷ OpenShift Container Platform リリースバージョンを定義します。
- ❸ RHACM バージョンを定義します。
- ❹ MCE バージョンを定義します。
- ❺ ディスク上のイメージをダウンロードするフォルダを定義します。
- ❻ オプション: 追加のイメージを保存するリポジトリを定義します。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。
- ❼ DU 設定に含まれる Operator の事前キャッシュを指定します。
- ❽ **--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 4.15.0 \
--acm-version 2.6.3 --mce-version 2.1.4 -f /mnt \
--img quay.io/custom/repository \
--du-profile -s \
--skip-imageset
```

関連情報

- オンラインの Red Hat レジストリーにアクセスするには、[OpenShift インストールカスタマイズツール](#) を参照してください。
- マルチクラスターエンジンの使用について、詳しくは [マルチクラスターエンジン Operator を使用したクラスターのライフサイクル](#) を参照してください。

14.5. GITOPS ZTP でのイメージの事前キャッシュ

SiteConfig マニフェストは、OpenShift クラスターをインストールおよび設定する方法を定義します。GitOps Zero Touch Provisioning (ZTP) プロビジョニングワークフローの場合、factory-precaching-cli ツールでは **SiteConfig** マニフェストに次の追加フィールドが必要です。

- **clusters.ignitionConfigOverride**
- **nodes.installerArgs**
- **nodes.ignitionConfigOverride**

追加フィールドを含む SiteConfig の例

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-5g-lab"
  namespace: "example-5g-lab"
spec:
  baseDomain: "example.domain.redhat.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "img4.9.10-x86-64-appsub" 1
  sshPublicKey: "ssh-rsa ..."
  clusters:
    - clusterName: "sno-worker-0"
      clusterImageSetNameRef: "eko4-img4.11.5-x86-64-appsub" 2
      clusterLabels:
        group-du-sno: ""
        common-411: true
        sites : "example-5g-lab"
        vendor: "OpenShift"
      clusterNetwork:
        - cidr: 10.128.0.0/14
          hostPrefix: 23
      machineNetwork:
        - cidr: 10.19.32.192/26
      serviceNetwork:
        - 172.30.0.0/16
      networkType: "OVNKubernetes"
      additionalNTPSources:
        - clock.corp.redhat.com
      ignitionConfigOverride: '{"ignition":{"version":"3.1.0"},"systemd":{"units":[{"name":"var-mnt.mount","enabled":true,"contents":"[Unit]\nDescription=Mount partition with artifacts\nBefore=precache-images.service\nBindsTo=precache-images.service\nStopWhenUnneeded=true\n\n[Mount]\nWhat=/dev/disk/by-partlabel/data\nWhere=/var/mnt\nType=xfs\nTimeoutSec=30\n\n[Install]\nRequiredBy=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\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%0AOCF_RELEASE_LIST%3D%22%24%7BOCF_RELEASE_LIST%3A-
```



```

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=dfs\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=k
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%0AOCF_RELEASE_LIST%3D%22%24%7BOCF_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%24OCF_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.tar.gz%0A%20%20if%20%5B%5B%20%24%3F%20-
eq%200%20%5D%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%5B%20%24%3F%20-
eq%200%20%5D%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%7BOCF_RELEASE_LI
ST%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

-l, --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。

Argo CD が監視している Git リポジトリに **SiteConfig** とオプションの **PolicyGenTemplates** カスタムリソース (CR) をアップロードすると、CR をハブクラスターと同期することで GitOps ZTP ワークフローを開始できます。

14.6. トラブルシューティング

14.6.1. Rendered catalog is invalid

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 4.15.0 --acm-version 2.5.4 \
```

```
--mce-version 2.0.4 -f /mnt \--img quay.io/custom/repository  
--du-profile -s --skip-imageset
```