



OpenShift Container Platform 4.13

スケーラビリティおよびパフォーマンス

実稼働環境における OpenShift Container Platform クラスターのスケーリングおよび
パフォーマンスチューニング

OpenShift Container Platform 4.13 スケーラビリティおよびパフォーマンス

実稼働環境における OpenShift Container Platform クラスターのスケーリングおよびパフォーマンスチューニング

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

本書では、クラスターをスケーリングし、OpenShift Container Platform 環境のパフォーマンスを最適化する方法について説明します。

目次

第1章 パフォーマンスとスケーラビリティの推奨プラクティス	5
1.1. コントロールプレーンの推奨プラクティス	5
1.2. インフラストラクチャーの推奨プラクティス	10
1.3. ETCD についての推奨されるプラクティス	13
第2章 オブジェクトの最大値に合わせた環境計画	24
2.1. メジャーリリースについての OPENSIFT CONTAINER PLATFORM のテスト済みクラスターの最大値	24
2.2. クラスターの最大値がテスト済みの OPENSIFT CONTAINER PLATFORM 環境および設定	27
2.3. テスト済みのクラスターの最大値に基づく環境計画	29
2.4. アプリケーション要件に合わせて環境計画を立てる方法	30
第3章 IBM Z & IBM(R) LINUXONE 環境で推奨されるホストの実践方法	34
3.1. CPU のオーバーコミットの管理	34
3.2. TRANSPARENT HUGE PAGES (THP) の無効	35
3.3. RECEIVE FLOW STEERING を使用したネットワークパフォーマンスの強化	35
3.4. ネットワーク設定の選択	36
3.5. Z/VM の HYPERPAV でディスクのパフォーマンスが高いことを確認します。	37
3.6. IBM Z ホストの RHEL KVM の推奨事項	38
第4章 NODE TUNING OPERATOR の使用	42
4.1. NODE TUNING OPERATOR について	42
4.2. NODE TUNING OPERATOR 仕様サンプルへのアクセス	42
4.3. クラスターに設定されるデフォルトのプロファイル	43
4.4. TUNED プロファイルが適用されていることの確認	44
4.5. カスタムチューニング仕様	45
4.6. カスタムチューニングの例	49
4.7. サポートされている TUNED デーモンプラグイン	51
4.8. ホステッドクラスターにおけるノードのチューニング設定	52
4.9. カーネルブートパラメーターを設定することによる、ホストされたクラスターの高度なノードチューニング	55
第5章 CPU マネージャーおよび TOPOLOGY MANAGER の使用	59
5.1. CPU マネージャーの設定	59
5.2. TOPOLOGY MANAGER ポリシー	63
5.3. TOPOLOGY MANAGER のセットアップ	64
5.4. POD の TOPOLOGY MANAGER ポリシーとの対話	64
第6章 NUMA 対応ワークロードのスケジューリング	66
6.1. NUMA 対応のスケジューリングについて	66
6.2. NUMA RESOURCES OPERATOR のインストール	68
6.3. NUMA 対応ワークロードのスケジューリング	70
6.4. 手動でのパフォーマンス設定による NUMA 対応ワークロードのスケジューリング	77
6.5. オプション: NUMA リソース更新のポーリング操作の設定	84
6.6. NUMA 対応スケジューリングのトラブルシューティング	86
第7章 スケーラビリティとパフォーマンスの最適化	96
7.1. ストレージの最適化	96
7.2. ルーティングの最適化	101
7.3. ネットワークの最適化	105
7.4. マウント NAMESPACE のカプセル化による CPU 使用率の最適化	107
第8章 ベアメタルホストの管理	115
8.1. ベアメタルホストおよびノードについて	115

8.2. ベアメタルホストのメンテナンス	115
第9章 BARE METAL EVENT RELAY を使用したベアメタルイベントのモニタリング	120
9.1. ベアメタル イベント	120
9.2. ベアメタルイベントの仕組み	120
9.3. AMQ メッセージングバスのインストール	124
9.4. クラスタードの REDFISH BMC ベアメタルイベントのサブスクリプション	125
9.5. ベアメタルイベント REST API リファレンスへのアプリケーションのサブスクリプション	131
9.6. PTP またはベアメタルイベントに HTTP トランスポートを使用するためのコンシューマーアプリケーションの移行	134
第10章 HUGE PAGE の機能およびそれらがアプリケーションによって消費される仕組み	136
10.1. HUGE PAGE の機能	136
10.2. HUGE PAGE がアプリケーションによって消費される仕組み	136
10.3. DOWNWARD API を使用した HUGE PAGE リソースの使用	137
10.4. 起動時の HUGE PAGE 設定	139
10.5. TRANSPARENT HUGE PAGES (THP) の無効化	141
第11章 低遅延チューニング	142
11.1. 低レイテンシーについて	142
11.2. リアルタイムおよび低レイテンシーワークロードのプロビジョニング	144
11.3. パフォーマンスプロファイルによる低レイテンシーを実現するためのノードのチューニング	154
11.4. NODE TUNING OPERATOR を使用した NIC キューの削減	168
11.5. 低レイテンシー CNF チューニングステータスのデバッグ	175
11.6. RED HAT サポート向けの低レイテンシーのチューニングデバッグデータの収集	177
第12章 プラットフォーム検証のためのレイテンシーテストの実行	180
12.1. レイテンシーテストを実行するための前提条件	180
12.2. レイテンシーテストの検出モードについて	180
12.3. レイテンシーの測定	181
12.4. レイテンシーテストの実行	182
12.5. レイテンシーテストの失敗レポートの生成	192
12.6. JUNIT レイテンシーテストレポートの生成	192
12.7. 単一ノードの OPENSIFT クラスタでレイテンシーテストを実行する	193
12.8. 切断されたクラスタでのレイテンシーテストの実行	193
12.9. CNF-TESTS コンテナでのエラーのトラブルシューティング	196
第13章 ワーカーレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスタの安定性の向上	197
13.1. ワーカーレイテンシープロファイルについて	197
13.2. クラスタ作成時にワーカー遅延プロファイルを実装する	200
13.3. ワーカーレイテンシープロファイルの使用と変更	201
13.4. WORKERLATENCYPROFILE の結果の値を表示する手順の例	203
第14章 パフォーマンスプロファイルの作成	205
14.1. PERFORMANCE PROFILE CREATOR の概要	205
14.2. パフォーマンスプロファイルの参照	217
14.3. 関連情報	218
第15章 ワークロードの分割	219
第16章 NODE OBSERVABILITY OPERATOR を使用した CRI-O および KUBELET プロファイリングデータのリスト	223
16.1. NODE OBSERVABILITY OPERATOR のワークフロー	223
16.2. NODE OBSERVABILITY OPERATOR のインストール	223
16.3. NODE OBSERVABILITY カスタムリソースの作成	226

16.4. プロファイリングクエリーの実行	227
第17章 ネットワーク遠端のクラスター	229
17.1. ネットワークファー遠端の課題	229
17.2. ZTP 用のハブクラスターの準備	234
17.3. 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	245
17.4. 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	262
17.5. 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	276
17.6. 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	288
17.7. 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	317
17.8. 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	332
17.9. 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	335
17.10. TOPOLOGY AWARE LIFECYCLE MANAGER を使用したマネージドクラスターの更新	367
17.11. 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	414
17.12. GITOPS ZTP の更新	430
17.13. GITOPS ZTP を使用した単一ノードの OPENSIFT クラスターの拡張	435
17.14. 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	443

第1章 パフォーマンスとスケーラビリティの推奨プラクティス

1.1. コントロールプレーンの推奨プラクティス

このトピックでは、OpenShift Container Platform のコントロールプレーンに関するパフォーマンスとスケーラビリティの推奨プラクティスについて説明します。

1.1.1. クラスターのスケーリングに関する推奨プラクティス

本セクションのガイダンスは、クラウドプロバイダーの統合によるインストールにのみ関連します。

以下のベストプラクティスを適用して、OpenShift Container Platform クラスター内のワーカーマシンの数をスケーリングします。ワーカーのマシンセットで定義されるレプリカ数を増やしたり、減らしたりしてワーカーマシンをスケーリングします。

クラスターをノード数のより高い値にスケールアップする場合:

- 高可用性を確保するために、ノードを利用可能なすべてのゾーンに分散します。
- 1度に 25 未満のマシンごとに 50 マシンまでスケールアップします。
- 定期的なプロバイダーの容量関連の制約を軽減するために、同様のサイズの別のインスタンスタイプを使用して、利用可能なゾーンごとに新規のコンピューティングマシンセットを作成することを検討してください。たとえば、AWS で、m5.large および m5d.large を使用します。



注記

クラウドプロバイダーは API サービスのクォータを実装する可能性があります。そのため、クラスターは段階的にスケーリングします。

コンピューティングマシンセットのレプリカが1度に高い値に設定される場合に、コントローラーはマシンを作成できなくなる可能性があります。OpenShift Container Platform が上部にデプロイされているクラウドプラットフォームが処理できる要求の数はプロセスに影響を与えます。コントローラーは、該当するステータスのマシンの作成、確認、および更新を試行する間に、追加のクエリーを開始します。OpenShift Container Platform がデプロイされるクラウドプラットフォームには API 要求の制限があり、過剰なクエリーが生じると、クラウドプラットフォームの制限によりマシンの作成が失敗する場合があります。

大規模なノード数にスケーリングする際にマシンヘルスチェックを有効にします。障害が発生する場合、ヘルスチェックは状態を監視し、正常でないマシンを自動的に修復します。



注記

大規模で高密度のクラスターをノード数を減らしてスケールダウンする場合には、長い時間がかかる可能性があります。このプロセスで、終了するノードで実行されているオブジェクトのドレイン (解放) またはエビクトが並行して実行されるためです。また、エビクトするオブジェクトが多過ぎる場合に、クライアントはリクエストのスロットリングを開始する可能性があります。デフォルトの1秒あたりのクライアントクエリー数 (QPS) とバーストレートは、現在それぞれ **5** と **10** に設定されています。これらの値は、OpenShift Container Platform では変更できません。

1.1.2. コントロールプレーンノードのサイジング

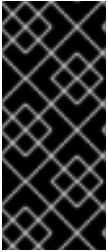
コントロールプレーンノードのリソース要件は、クラスター内のノードとオブジェクトの数とタイプによって異なります。次のコントロールプレーンノードサイズの推奨事項は、コントロールプレーン密度に焦点を当てたテストまたは **クラスター密度** の結果に基づいています。このテストでは、指定された数の namespace にわたって次のオブジェクトを作成します。

- 1 イメージストリーム
- 1 ビルド
- 5 つのデプロイメント、**sleep** 状態の 2 つの Pod レプリカ、4 つのシークレット、4 つの config map、およびそれぞれ 1 つの下位 API ボリュームのマウント
- 5 つのサービス。それぞれが以前のデプロイメントの 1 つの TCP/8080 および TCP/8443 ポートを指します。
- 以前のサービスの最初を指す 1 つのルート
- 2048 個のランダムな文字列文字を含む 10 個のシークレット
- 2048 個のランダムな文字列文字を含む 10 個の config map

ワーカーノードの数	クラスター密度 (namespace)	CPU コア数	メモリー (GB)
24	500	4	16
120	1000	8	32
252	4000	16、ただし OVN-Kubernetes ネットワークプラグインを使用する場合は 24	64、ただし OVN-Kubernetes ネットワークプラグインを使用する場合は 128
501、ただし OVN-Kubernetes ネットワークプラグインではテストされていません	4000	16	96

上の表のデータは、r5.4xlarge インスタンスをコントロールプレーンノードとして使用し、m5.2xlarge インスタンスをワーカーノードとして使用する、AWS 上で実行される OpenShift Container Platform をベースとしています。

3 つのコントロールプレーンノードがある大規模で高密度のクラスターでは、いずれかのノードが停止、起動、または障害が発生すると、CPU とメモリーの使用量が急上昇します。障害は、電源、ネットワーク、または基礎となるインフラストラクチャーの予期しない問題、またはコストを節約するためにシャットダウンした後、クラスターが再起動する意図的なケースが原因である可能性があります。残りの 2 つのコントロールプレーンノードは、高可用性を維持するために負荷を処理する必要があります。これにより、リソースの使用量が増えます。これは、コントロールプレーンモードが遮断 (cordon)、ドレイン (解放) され、オペレーティングシステムおよびコントロールプレーン Operator の更新を適用するために順次再起動されるため、アップグレード時に想定される動作になります。障害が繰り返し発生しないようにするには、コントロールプレーンノードでの全体的な CPU およびメモリーリソース使用状況を、利用可能な容量の最大 60% に維持し、使用量の急増に対応できるようにします。リソース不足による潜在的なダウンタイムを回避するために、コントロールプレーンノードの CPU およびメモリーを適宜増やします。



重要

ノードのサイジングは、クラスター内のノードおよびオブジェクトの数によって異なります。また、オブジェクトがそのクラスター上でアクティブに作成されるかどうかによっても異なります。オブジェクトの作成時に、コントロールプレーンは、オブジェクトが **running** フェーズにある場合と比較し、リソースの使用状況においてよりアクティブな状態になります。

Operator Lifecycle Manager (OLM) はコントロールプレーンノードで実行され、OLM のメモリーフットプリントは OLM がクラスター上で管理する必要のある namespace およびユーザーによってインストールされる Operator の数によって異なります。OOM による強制終了を防ぐには、コントロールプレーンノードのサイズを適切に設定する必要があります。以下のデータポイントは、クラスター最大のテストの結果に基づいています。

namespace 数	アイドル状態の OLM メモリー (GB)	ユーザー Operator が 5 つインストールされている OLM メモリー (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6



重要

以下の設定でのみ、実行中の OpenShift Container Platform 4.13 クラスターでコントロールプレーンのノードサイズを変更できます。

- ユーザーがプロビジョニングしたインストール方法でインストールされたクラスター。
- installer-provisioned infrastructure インストール方法でインストールされた AWS クラスター。
- コントロールプレーンマシンセットを使用してコントロールプレーンマシンを管理するクラスター。

他のすべての設定では、合計ノード数を見積もり、インストール時に推奨されるコントロールプレーンノードサイズを使用する必要があります。



重要

この推奨事項は、ネットワークプラグインとして OpenShift SDN を使用して OpenShift Container Platform クラスターでキャプチャーされたデータポイントに基づいています。



注記

OpenShift Container Platform 4.13 では、OpenShift Container Platform 3.11 以前のバージョンと比較すると、CPU コア (500 ミリコア) の半分がデフォルトでシステムによって予約されるようになりました。サイズはこれを考慮に入れて決定されます。

1.1.2.1. コントロールプレーンマシン用により大きな Amazon Web Services インスタンスタイプを選択する

Amazon Web Services (AWS) クラスター内のコントロールプレーンマシンがより多くのリソースを必要とする場合は、コントロールプレーンマシンが使用するより大きな AWS インスタンスタイプを選択できます。



注記

コントロールプレーンマシンセットを使用するクラスターの手順は、コントロールプレーンマシンセットを使用しないクラスターの手順とは異なります。

クラスター内の **ControlPlaneMachineSet** CR の状態が不明な場合は、[CR の状態を確認](#)できます。

1.1.2.1.1. コントロールプレーンマシンセットを使用して Amazon Web Services インスタンスタイプを変更する

コントロールプレーンマシンセットのカスタムリソース (CR) の仕様を更新することで、コントロールプレーンマシンが使用する Amazon Web Services (AWS) インスタンスタイプを変更できます。

前提条件

- AWS クラスターは、コントロールプレーンマシンセットを使用します。

手順

1. 次のコマンドを実行して、コントロールプレーンマシンセットの CR を編集します。

```
$ oc --namespace openshift-machine-api edit controlplanemachineset.machine.openshift.io cluster
```

2. **providerSpec** フィールドの下で以下の行を編集します。

```
providerSpec:
  value:
    ...
    instanceType: <compatible_aws_instance_type> ❶
```

- ❶ 前の選択と同じベースで、より大きな AWS インスタンスタイプを指定します。たとえば、**m6i.xlarge** を **m6i.2xlarge** または **m6i.4xlarge** に変更できます。

3. 変更を保存します。

- デフォルトの **RollingUpdate** 更新戦略を使用するクラスターの場合、Operator は自動的に変更をコントロールプレーン設定に伝達します。
- **OnDelete** 更新戦略を使用するように設定されているクラスターの場合、コントロールプレーンマシンを手動で置き換える必要があります。

関連情報

- [コントロールプレーンマシンセットを使用したコントロールプレーンマシンの管理](#)

1.1.2.1.2. AWS コンソールを使用して Amazon Web Services インスタンスタイプを変更する

AWS コンソールでインスタンスタイプを更新することにより、コントロールプレーンマシンが使用するアマゾンウェブサービス (AWS) インスタンスタイプを変更できます。

前提条件

- クラスターの EC2 インスタンスを変更するために必要なアクセス許可を持つ AWS コンソールにアクセスできます。
- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできます。

手順

1. AWS コンソールを開き、コントロールプレーンマシンのインスタンスを取得します。
2. コントロールプレーンマシンインスタンスを1つ選択します。
 - a. 選択したコントロールプレーンマシンについて、etcd スナップショットを作成して etcd データをバックアップします。詳細については、etcd のバックアップを参照してください。
 - b. AWS コンソールで、コントロールプレーンマシンインスタンスを停止します。

- c. 停止したインスタンスを選択し、**Actions** → **Instance Settings** → **Change instance type** をクリックします。
 - d. インスタンスをより大きなタイプに変更し、タイプが前の選択と同じベースであることを確認して、変更を適用します。たとえば、**m6i.xlarge** を **m6i.2xlarge** または **m6i.4xlarge** に変更できます。
 - e. インスタンスを起動します。
 - f. OpenShift Container Platform クラスターにインスタンスに対応する **Machine** オブジェクトがある場合、AWS コンソールで設定されたインスタンスタイプと一致するようにオブジェクトのインスタンスタイプを更新します。
3. コントロールプレーンマシンごとにこのプロセスを繰り返します。

関連情報

- [etcd のバックアップ](#)
- [インスタンスタイプの変更に関する AWS ドキュメント](#)

1.2. インフラストラクチャーの推奨プラクティス

このトピックでは、OpenShift Container Platform のインフラストラクチャーに関するパフォーマンスとスケーラビリティの推奨プラクティスについて説明します。

1.2.1. インフラストラクチャーノードのサイジング

インフラストラクチャーノード は、OpenShift Container Platform 環境の各部分を実行するようにラベル付けされたノードです。これらの要素により、Prometheus のメトリクスまたは時系列の数が増加する可能性があり、インフラストラクチャーノードのリソース要件はクラスターの使用年数、ノード、およびオブジェクトによって異なります。次のインフラストラクチャーノードサイズの推奨事項は、**コントロールプレーンノードのサイジング** セクションで詳しく説明されているクラスター密度テストで観察された結果に基づいています。モニタリングスタックとデフォルトの Ingress コントローラーは、これらのノードに移動されています。

ワーカーノードの数	クラスター密度または namespace の数	CPU コア数	メモリー (GB)
27	500	4	24
120	1000	8	48
252	4000	16	128
501	4000	32	128

通常、3つのインフラストラクチャーノードはクラスターごとに推奨されます。



重要

これらのサイジングの推奨事項は、ガイドラインとして使用する必要があります。Prometheus はメモリー集約型のアプリケーションであり、リソースの使用率はノード数、オブジェクト数、Prometheus メトリクスの収集間隔、メトリクスまたは時系列、クラスターの使用年数などのさまざまな要素によって異なります。さらに、ルーターのリソース使用量は、ルートの数とインバウンド要求の量/タイプによっても影響を受ける可能性があります。

これらの推奨事項は、クラスターの作成時にインストールされたモニタリング、イングレス、およびレジストリーインフラストラクチャーコンポーネントをホストするインフラストラクチャーノードにのみ適用されます。



注記

OpenShift Container Platform 4.13 では、OpenShift Container Platform 3.11 以前のバージョンと比較すると、CPU コア (500 ミリコア) の半分がデフォルトでシステムによって予約されるようになりました。これは、上記のサイジングの推奨内容に影響します。

1.2.2. Cluster Monitoring Operator のスケーリング

OpenShift Container Platform は、Cluster Monitoring Operator が収集し、Prometheus ベースのモニタリングスタックに保存するメトリクスを公開します。管理者は、**Observe** → **Dashboards** に移動して、OpenShift Container Platform Web コンソールでシステムリソース、コンテナ、およびコンポーネントメトリクスのダッシュボードを表示できます。

1.2.3. Prometheus データベースのストレージ要件

Red Hat は、スケールサイズに応じて各種のテストを実行しました。



注記

- 次の Prometheus ストレージ要件は規定されていないため、参考として使用してください。ワークロードのアクティビティーおよびリソースの密度に応じて、クラスターでより多くのリソース消費が観察される可能性があります。これには、Pod、コンテナ、ルート、Prometheus により収集されるメトリクスを公開する他のリソースの数が含まれます。
- ストレージ要件に合わせて、サイズベースのデータ保持ポリシーを設定できます。

表1.1 クラスター内のノード/Pod の数に基づく Prometheus データベースのストレージ要件

ノード数	Pod 数(Pod あたり 2 コンテナ)	1日あたりの Prometheus ストレージの増加量	15日ごとの Prometheus ストレージの増加量	ネットワーク (tsdb チャンクに基づく)
50	1800	6.3 GB	94 GB	16 MB
100	3600	13 GB	195 GB	26 MB
150	5400	19 GB	283 GB	36 MB

ノード数	Pod数(Podあたり 2コンテナ)	1日あたりの Prometheus スト レージの増加量	15日ごとの Prometheus スト レージの増加量	ネットワーク (tsdb チャンクに 基づく)
200	7200	25 GB	375 GB	46 MB

ストレージ要件が計算値を超過しないようにするために、オーバーヘッドとして予期されたサイズのおよそ 20% が追加されています。

上記の計算は、デフォルトの OpenShift Container Platform Cluster Monitoring Operator についての計算です。



注記

CPU の使用率による影響は大きくありません。比率については、およそ 50 ノードおよび 1800 Pod ごとに 1 コア (/40) になります。

OpenShift Container Platform についての推奨事項

- 3 つ以上のインフラストラクチャー (infra) ノードを使用します。
- Non-Volatile Memory Express (SSD または NVMe) ドライブを備えた少なくとも 3 つの `openshift-container-storage` ノードを使用します。

1.2.4. クラスターモニタリングの設定

クラスターモニタリングスタック内の Prometheus コンポーネントのストレージ容量を増やすことができます。

手順

Prometheus のストレージ容量を拡張するには、以下を実行します。

1. YAML 設定ファイル `cluster-monitoring-config.yml` を作成します。以下に例を示します。

```

apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} ❶
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: {{STORAGE_CLASS}} ❷
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} ❸
    alertmanagerMain:
      nodeSelector:
        node-role.kubernetes.io/infra: ""

```



```

volumeClaimTemplate:
  spec:
    storageClassName: {{STORAGE_CLASS}} 4
    resources:
      requests:
        storage: {{ALERTMANAGER_STORAGE_SIZE}} 5
  metadata:
    name: cluster-monitoring-config
    namespace: openshift-monitoring

```

- 1 Prometheus の保持のデフォルト値は **PROMETHEUS_RETENTION_PERIOD=15d** です。時間は、接尾辞 s、m、h、d のいずれかを使用する単位で測定されます。
- 2 4 クラスターのストレージクラス。
- 3 標準の値は **PROMETHEUS_STORAGE_SIZE=2000Gi** です。ストレージの値には、接尾辞 E、P、T、G、M、K のいずれかを使用した単純な整数または固定小数点整数を使用できます。また、2 のべき乗の値 (Ei、Pi、Ti、Gi、Mi、Ki) を使用することもできます。
- 5 標準の値は **ALERTMANAGER_STORAGE_SIZE=20Gi** です。ストレージの値には、接尾辞 E、P、T、G、M、K のいずれかを使用した単純な整数または固定小数点整数を使用できます。また、2 のべき乗の値 (Ei、Pi、Ti、Gi、Mi、Ki) を使用することもできます。

2. 保存期間、ストレージクラス、およびストレージサイズの値を追加します。
3. ファイルを保存します。
4. 以下を実行して変更を適用します。

```
$ oc create -f cluster-monitoring-config.yaml
```

1.2.5. 関連情報

- [OpenShift 4 のインフラストラクチャーノード](#)
- [OpenShift Container Platform クラスターの最大値](#)
- [インフラストラクチャーマシンセットの作成](#)

1.3. ETCD についての推奨されるプラクティス

このトピックでは、OpenShift Container Platform の etcd に関するパフォーマンスとスケーラビリティの推奨プラクティスについて説明します。

1.3.1. etcd についての推奨されるプラクティス

etcd はデータをディスクに書き込み、プロポーザルをディスクに保持するため、そのパフォーマンスはディスクのパフォーマンスに依存します。etcd は特に I/O を集中的に使用するわけではありませんが、最適なパフォーマンスと安定性を得るには、低レイテンシーのブロックデバイスが必要です。etcd のコンセンサスプロトコルは、メタデータをログ (WAL) に永続的に保存することに依存しているため、etcd はディスク書き込みの遅延に敏感です。遅いディスクと他のプロセスからのディスクアクティビティは、長い fsync 待ち時間を引き起こす可能性があります。

これらの待ち時間により、etcd はハートビートを見逃し、新しいプロポーザルを時間どおりにディスク

にコミットせず、最終的にリクエストのタイムアウトと一時的なリーダーの喪失を経験する可能性があります。書き込みレイテンシーが高いと、OpenShift API の速度も低下し、クラスターのパフォーマンスに影響します。これらの理由により、I/O を区別する、または集約型であり、同一基盤として I/O インフラストラクチャーを共有する他のワークロードをコントロールプレーンノードに併置することは避けてください。

レイテンシーに関しては、8000 バイト長の 50 IOPS 以上を連続して書き込むことができるブロックデバイス上で etcd を実行します。つまり、レイテンシーが 10 ミリ秒の場合、fdatsync を使用して WAL の各書き込みを同期することに注意してください。負荷の高いクラスターの場合、8000 バイト (2 ミリ秒) の連続 500 IOPS が推奨されます。これらの数値を測定するには、fio などのベンチマークツールを使用できます。

このようなパフォーマンスを実現するには、低レイテンシーで高スループットの SSD または NVMe ディスクに支えられたマシンで etcd を実行します。シングルレベルセル (SLC) ソリッドステートドライブ (SSD) を検討してください。これは、メモリーセルごとに 1 ビットを提供し、耐久性と信頼性が高く、書き込みの多いワークロードに最適です。



注記

etcd の負荷は、ノードや Pod の数などの静的要因と、Pod の自動スケーリング、Pod の再起動、ジョブの実行、その他のワークロード関連イベントが原因となるエンドポイントの変更などの動的要因から生じます。etcd セットアップのサイズを正確に設定するには、ワークロードの具体的な要件を分析する必要があります。etcd の負荷に影響を与えるノード、Pod、およびその他の関連要素の数を考慮してください。

最適な etcd パフォーマンスを得るには、ハードドライブで以下を適用します。

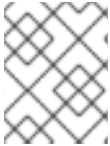
- 専用の etcd ドライブを使用します。iSCSI などのネットワーク経由で通信するドライブは回避します。etcd ドライブにログファイルやその他の重いワークロードを配置しないでください。
- 読み取りおよび書き込みを高速化するために、低レイテンシードライブを優先的に使用します。
- 圧縮と最適化を高速化するために、高帯域幅の書き込みを優先的に使用します。
- 障害からの回復を高速化するために、高帯域幅の読み取りを優先的に使用します。
- 最小の選択肢としてソリッドステートドライブを使用します。実稼働環境には NVMe ドライブの使用が推奨されます。
- 高い信頼性を確保するためには、サーバーグレードのハードウェアを使用します。



注記

NAS または SAN のセットアップ、および回転するドライブは避けてください。Ceph Rados Block Device (RBD) およびその他のタイプのネットワーク接続ストレージでは、予測できないネットワーク遅延が発生する可能性があります。etcd ノードに大規模な高速ストレージを提供するには、PCI パススルーを使用して NVMe デバイスをノードに直接渡します。

fio などのユーティリティを使用して、常にベンチマークを行ってください。このようなユーティリティを使用すると、クラスターのパフォーマンスが向上するにつれて、そのパフォーマンスを継続的に監視できます。



注記

ネットワークファイルシステム (NFS) プロトコルまたはその他のネットワークベースのファイルシステムの使用は避けてください。

デプロイされた OpenShift Container Platform クラスターでモニターする主要なメトリクスの一部は、etcd ディスクの write ahead log 期間の p99 と etcd リーダーの変更数です。Prometheus を使用してこれらのメトリクスを追跡します。



注記

etcd メンバーデータベースのサイズは、通常の運用時にクラスター内で異なる場合があります。この違いは、リーダーのサイズが他のメンバーと異なっても、クラスターのアップグレードには影響しません。

OpenShift Container Platform クラスターの作成前または作成後に etcd のハードウェアを検証するには、`fio` を使用できます。

前提条件

- Podman や Docker などのコンテナランタイムは、テストしているマシンにインストールされます。
- データは `/var/lib/etcd` パスに書き込まれます。

手順

- `fio` を実行し、結果を分析します。
 - Podman を使用する場合は、次のコマンドを実行します。


```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```
 - Docker を使用する場合は、次のコマンドを実行します。


```
$ sudo docker run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

この出力では、実行からキャプチャーされた `fsync` メトリクスの 99 パーセントの比較でディスクが 10 ms 未満かどうかを確認して、ディスクの速度が etcd をホストするのに十分であるかどうかを報告します。I/O パフォーマンスの影響を受ける可能性のある最も重要な etcd メトリックのいくつかを以下に示します。

- `etcd_disk_wal_fsync_duration_seconds_bucket` メトリックは、etcd の WAL `fsync` 期間を報告します。
- `etcd_disk_backend_commit_duration_seconds_bucket` メトリックは、etcd バックエンドコミットの待機時間を報告します。
- `etcd_server_leader_changes_seen_total` メトリックは、リーダーの変更を報告します。

etcd はすべてのメンバー間で要求を複製するため、そのパフォーマンスはネットワーク入出力 (I/O) のレイテンシーによって大きく変わります。ネットワークのレイテンシーが高くなると、etcd のハートビートの時間は選択のタイムアウトよりも長くなり、その結果、クラスターに中断をもたらすリーダー

の選択が発生します。デプロイされた OpenShift Container Platform クラスターでのモニターの主要なメトリクスは、各 etcd クラスターメンバーの etcd ネットワークピアレイテンシーの 99 番目のパーセンタイルになります。Prometheus を使用してメトリクスを追跡します。

`histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[2m]))` メトリックは、etcd がメンバー間でクライアントリクエストの複製を完了するまでのラウンドトリップ時間をレポートします。50 ミリ秒未満であることを確認してください。

関連情報

- [fio を使用して OpenShiftContainerPlatform で OpenShift Container Platform ディスクのパフォーマンスを確認する方法](#)
- [OpenShift Container Platform の etcd パフォーマンスに関するトラブルシューティングガイド](#)

1.3.2. etcd を別のディスクに移動する

etcd を共有ディスクから別のディスクに移動して、パフォーマンスの問題を防止または解決できます。

Machine Config Operator (MCO) は、OpenShift Container Platform 4.15 コンテナストレージのセカンドリーディスクをマウントします。



注記

この手順では、`/var/` などのルートファイルシステムの一部を、インストール済みノードの別のディスクまたはパーティションに移動しません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。
- **MachineConfigPool** は `metadata.labels.machineconfiguration.openshift.io/role` と一致する必要があります。これは、コントローラー、ワーカー、またはカスタムプールに適用されます。

手順

1. 新しいディスクをクラスターに接続し、デバッグシェルで **lsblk** コマンドを使用して、ディスクがノード内で検出されることを確認します。

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

```
# lsblk
```

lsblk コマンドで報告された新しいディスクのデバイス名をメモします。

2. 次のような内容を含む **MachineConfig** YAML ファイルを **etcd-mc.yml** という名前で作成し、`<new_disk_name>` のインスタンスをメモしたデバイス名に置き換えます。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
```

```
labels:
  machineconfiguration.openshift.io/role: master
name: 98-var-lib-etcd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Make File System on /dev/<new_disk_name>
            DefaultDependencies=no
            BindsTo=dev-<new_disk_name>.device
            After=dev-<new_disk_name>.device var.mount
            Before=systemd-fsck@dev-<new_disk_name>.service

            [Service]
            Type=oneshot
            RemainAfterExit=yes
            ExecStart=/usr/lib/systemd/systemd-makefs xfs /dev/<new_disk_name>
            TimeoutSec=0

            [Install]
            WantedBy=var-lib-containers.mount
          enabled: true
          name: systemd-mkfs@dev-<new_disk_name>.service
        - contents: |
            [Unit]
            Description=Mount /dev/<new_disk_name> to /var/lib/etcd
            Before=local-fs.target
            Requires=systemd-mkfs@dev-<new_disk_name>.service
            After=systemd-mkfs@dev-<new_disk_name>.service var.mount

            [Mount]
            What=/dev/<new_disk_name>
            Where=/var/lib/etcd
            Type=xfs
            Options=defaults,prjquota

            [Install]
            WantedBy=local-fs.target
          enabled: true
          name: var-lib-etcd.mount
        - contents: |
            [Unit]
            Description=Sync etcd data if new mount is empty
            DefaultDependencies=no
            After=var-lib-etcd.mount var.mount
            Before=crio.service

            [Service]
            Type=oneshot
            RemainAfterExit=yes
            ExecCondition=/usr/bin/test ! -d /var/lib/etcd/member
            ExecStart=semanage fcontext -a -e /sysroot/ostree/deploy/rhcos/var/lib/etcd/
```

```

/var/lib/etcd/
  ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
  TimeoutSec=0

  [Install]
  WantedBy=multi-user.target graphical.target
  enabled: true
  name: sync-var-lib-etcd-to-etcd.service
- contents: |
  [Unit]
  Description=Restore recursive SELinux security contexts
  DefaultDependencies=no
  After=var-lib-etcd.mount
  Before=crio.service

  [Service]
  Type=oneshot
  RemainAfterExit=yes
  ExecStart=/sbin/restorecon -R /var/lib/etcd/
  TimeoutSec=0

  [Install]
  WantedBy=multi-user.target graphical.target
  enabled: true
  name: restorecon-var-lib-etcd.service

```

3. **cluster-admin** 権限を持つユーザーとしてクラスターにログインし、マシン設定を作成します。

```
$ oc login -u <username> -p <password>
```

```
$ oc create -f etcd-mc.yml
```

ノードが更新され、再起動されます。再起動が完了すると、次のイベントが発生します。

- 指定したディスクに XFS ファイルシステムが作成されます。
 - ディスクは **/var/lib/etc** にマウントされます。
 - **/sysroot/ostree/deploy/rhcos/var/lib/etcd** のコンテンツは **/var/lib/etcd** に同期されます。
 - **/var/lib/etcd** の **SELinux** ラベルの復元が強制されます。
 - 古いコンテンツは削除されません。
4. ノードが別のディスク上に配置されたら、次のような内容で **etcd-mc.yml** ファイルを更新し、**<new_disk_name>** のインスタンスをメモしたデバイス名に置き換えます。

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:

```

```

config:
  ignition:
    version: 3.2.0
  systemd:
    units:
      - contents: |
          [Unit]
          Description=Mount /dev/<new_disk_name> to /var/lib/etcd
          Before=local-fs.target
          After=var.mount

          [Mount]
          What=/dev/<new_disk_name>
          Where=/var/lib/etcd
          Type=xf
          Options=defaults,prjquota

          [Install]
          WantedBy=local-fs.target
        enabled: true
        name: var-lib-etcd.mount

```

5. ノードの再起動を防止するため、デバイスの作成と同期に使用するロジックを削除する修正バージョンを適用します。

```
$ oc replace -f etcd-mc.yml
```

検証手順

- ノードのデバッグシェルで `grep <new_disk_name>/proc/mounts` コマンドを実行して、ディスクがマウントされていることを確認します。

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

```
# grep <new_disk_name> /proc/mounts
```

出力例

```
/dev/nvme1n1 /var/lib/etcd xfs
rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,prjquota 0 0
```

関連情報

- [Red Hat Enterprise Linux CoreOS \(RHCOS\)](#)

1.3.3. etcd データのデフラグ

大規模で密度の高いクラスターの場合に、キースペースが過剰に拡大し、スペースのクォータを超過すると、etcd は低下するパフォーマンスの影響を受ける可能性があります。etcd を定期的に維持および最適化して、データストアのスペースを解放します。Prometheus で etcd メトリックをモニターし、必要に応じてデフラグします。そうしないと、etcd はクラスター全体のアラームを発生させ、クラスターをメンテナンスモードにして、キーの読み取りと削除のみを受け入れる可能性があります。

これらの主要な指標をモニターします。

- **etcd_server_quota_backend_bytes**、これは現在のクォータ制限です
- **etcd_mvcc_db_total_size_in_use_in_bytes**、これはヒストリーコンパクション後の実際のデータベース使用状況を示します。
- **etcd_mvcc_db_total_size_in_bytes** はデフラグ待ちの空き領域を含むデータベースサイズを表します。

etcd データをデフラグし、etcd 履歴の圧縮などのディスクの断片化を引き起こすイベント後にディスク領域を回収します。

履歴の圧縮は 5 分ごとに自動的に行われ、これによりバックエンドデータベースにギャップが生じます。この断片化された領域は etcd が使用できますが、ホストファイルシステムでは利用できません。ホストファイルシステムでこの領域を使用できるようにするには、etcd をデフラグする必要があります。

デフラグは自動的に行われますが、手動でトリガーすることもできます。



注記

etcd Operator はクラスター情報を使用してユーザーの最も効率的な操作を決定するため、ほとんどの場合、自動デフラグが適しています。

1.3.3.1. 自動デフラグ

etcd Operator はディスクを自動的にデフラグします。手動による介入は必要ありません。

以下のログのいずれかを表示して、デフラグプロセスが成功したことを確認します。

- etcd ログ
- cluster-etcd-operator Pod
- Operator ステータスのエラーログ



警告

自動デフラグにより、Kubernetes コントローラマネージャーなどのさまざまな OpenShift コアコンポーネントでリーダー選出の失敗が発生し、失敗したコンポーネントの再起動がトリガーされる可能性があります。再起動は無害であり、次に実行中のインスタンスへのフェイルオーバーをトリガーするか、再起動後にコンポーネントが再び作業を再開します。

最適化が成功した場合のログ出力の例

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

最適化に失敗した場合のログ出力の例

failed defrag on member: <member_name>, memberID: <member_id>: <error_message>

1.3.3.2. 手動デフラグ

Prometheus アラートは、手動でのデフラグを使用する必要がある場合を示します。アラートは次の 2 つの場合に表示されます。

- etcd が使用可能なスペースの 50% 以上を 10 分を超過して使用する場合
- etcd が合計データベースサイズの 50% 未満を 10 分を超過してアクティブに使用している場合

また、PromQL 式を使用した最適化によって解放される etcd データベースのサイズ (MB 単位) を確認することで、最適化が必要かどうかを判断することもできます ((`etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes`)/1024/1024)。



警告

etcd のデフラグはプロセスを阻止するアクションです。etcd メンバーはデフラグが完了するまで応答しません。このため、各 Pod のデフラグアクションごとに少なくとも 1 分間待機し、クラスターが回復できるようにします。

以下の手順に従って、各 etcd メンバーで etcd データをデフラグします。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. リーダーを最後にデフラグする必要があるため、どの etcd メンバーがリーダーであるかを判別します。
 - a. etcd Pod のリストを取得します。

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

出力例

```
etcd-ip-10-0-159-225.example.redhat.com      3/3   Running   0      175m
10.0.159.225 ip-10-0-159-225.example.redhat.com <none> <none>
etcd-ip-10-0-191-37.example.redhat.com      3/3   Running   0      173m
10.0.191.37 ip-10-0-191-37.example.redhat.com <none> <none>
etcd-ip-10-0-199-170.example.redhat.com     3/3   Running   0      176m
10.0.199.170 ip-10-0-199-170.example.redhat.com <none> <none>
```

- b. Pod を選択し、以下のコマンドを実行して、どの etcd メンバーがリーダーであるかを判別します。

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint
status --cluster -w table
```

出力例

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
```

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|   ENDPOINT   |   ID   | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.5.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
```

この出力の **IS LEADER** 列に基づいて、**https://10.0.199.170:2379** エンドポイントがリーダーになります。このエンドポイントを直前の手順の出力に一致させると、リーダーの Pod 名は **etcd-ip-10-0-199-170.example.redhat.com** になります。

2. etcd メンバーのデフラグ。

- a. 実行中の etcd コンテナに接続し、リーダーでは **ない** Pod の名前を渡します。

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
```

- b. **ETCDCTL_ENDPOINTS** 環境変数の設定を解除します。

```
sh-4.4# unset ETCDCTL_ENDPOINTS
```

- c. etcd メンバーのデフラグを実行します。

```
sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
```

出力例

```
Finished defragmenting etcd member[https://localhost:2379]
```

タイムアウトエラーが発生した場合は、コマンドが正常に実行されるまで **--command-timeout** の値を増やします。

- d. データベースサイズが縮小されていることを確認します。

```
sh-4.4# etcdctl endpoint status -w table --cluster
```

出力例

```

+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|   ENDPOINT   |   ID   | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.5.9 | 41 MB | false | false |
7 | 91624 | 91624 | | ①
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.5.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+

```

この例では、この etcd メンバーのデータベースサイズは、開始時のサイズの 104 MB ではなく 41 MB です。

- e. これらの手順を繰り返して他の etcd メンバーのそれぞれに接続し、デフラグします。常に最後にリーダーをデフラグします。
etcd Pod が回復するように、デフラグアクションごとに 1分以上待機します。etcd Pod が回復するまで、etcd メンバーは応答しません。
3. 領域のクォータの超過により **NOSPACE** アラームがトリガーされる場合、それらをクリアします。
 - a. **NOSPACE** アラームがあるかどうかを確認します。

```
sh-4.4# etcdctl alarm list
```

出力例

```
memberID:12345678912345678912 alarm:NOSPACE
```

- b. アラームをクリアします。

```
sh-4.4# etcdctl alarm disarm
```

第2章 オブジェクトの最大値に合わせた環境計画

OpenShift Container Platform クラスターの計画時に以下のテスト済みのオブジェクトの最大値を考慮します。

これらのガイドラインは、最大規模のクラスターに基づいています。小規模なクラスターの場合、最大値はこれより低くなります。指定のしきい値に影響を与える要因には、etcd バージョンやストレージデータ形式などの多数の要因があります。

ほとんど場合、これらの制限値を超えると、パフォーマンスが全体的に低下します。ただし、これによって必ずしもクラスターに障害が発生する訳ではありません。



警告

Pod の起動および停止が多数あるクラスターなど、急速な変更が生じるクラスターは、実質的な最大サイズが記録よりも小さくなる可能性があります。

2.1. メジャーリリースについての OPENSIFT CONTAINER PLATFORM のテスト済みクラスターの最大値



注記

Red Hat は、OpenShift Container Platform クラスターのサイズ設定に関する直接的なガイダンスを提供していません。これは、クラスターが OpenShift Container Platform のサポート範囲内にあるかどうかを判断するには、クラスターのスケールを制限するすべての多次元な要因を慎重に検討する必要があります。

OpenShift Container Platform は、クラスターの絶対最大値ではなく、テスト済みのクラスター最大値をサポートします。OpenShift Container Platform のバージョン、コントロールプレーンのワークロード、およびネットワークプラグインのすべての組み合わせがテストされているわけではないため、以下の表は、すべてのデプロイメントの規模の絶対的な期待値を表すものではありません。すべてのディメンションを同時に最大にスケールリングすることはできない場合があります。この表には、特定のワークロードとデプロイメント設定に対してテストされた最大値が含まれており、同様のデプロイメントで何が期待できるかについてのスケールガイドとして機能します。

最大値のタイプ	4.x テスト済みの最大値
ノード数	2,000 ^[1]
Pod の数 ^[2]	150,000
ノードあたりの Pod 数	2,500 ^{[3][4]}
コアあたりの Pod 数	デフォルト値はありません。

最大値のタイプ	4.x テスト済みの最大値
namespace の数 ^[5]	10,000
ビルド数	10,000(デフォルト Pod RAM 512 Mi)- Source-to-Image (S2I) ビルドストラテジー
namespace ごとの Pod の数 ^[6]	25,000
Ingress Controller ごとのルートとバックエンドの数	ルーターあたり 2,000
シークレットの数	80,000
config map の数	90,000
サービスの数 ^[7]	10,000
namespace ごとのサービス数	5,000
サービスごとのバックエンド数	5,000
namespace ごとのデプロイメントの数 ^[6]	2,000
ビルド設定の数	12,000
カスタムリソース定義 (CRD) の数	512 ^[8]

1. 一時停止 Pod は、2000 ノードスケールで OpenShift Container Platform のコントロールプレーンコンポーネントにストレスをかけるためにデプロイされました。同様の数値にスケールリングできるかどうかは、特定のデプロイメントとワークロードのパラメーターによって異なります。
2. ここで表示される Pod 数はテスト用の Pod 数です。実際の Pod 数は、アプリケーションのメモリ、CPU、およびストレージ要件により異なります。
3. これは、31 台のサーバー (3 つのコントロールプレーン、2 つのインフラストラクチャーノード、および 26 のワーカーノード) を備えたクラスターでテストされました。2,500 のユーザー Pod が必要な場合は、各ノードが 2000 超の Pod を内包できる規模のネットワークを割り当てるために **hostPrefix** を **20** に設定し、カスタム kubelet 設定で **maxPods** を **2500** に設定する必要があります。詳細は、[OCP 4.13 でノードごとに 2500 Pod を実行する](#) を参照してください。
4. **OVNKubernetes** ネットワークプラグインを使用するクラスターの場合、ノードごとにテストされる最大 Pod 数は 2,500 です。**OpenShiftSDN** ネットワークプラグインのノードごとにテストされる最大 Pod 数は 500 Pod です。
5. 有効なプロジェクトが多数ある場合、キースペースが過剰に拡大し、スペースのクォータを超過すると、etcd はパフォーマンスの低下による影響を受ける可能性があります。etcd ストレージを解放するために、デフラグを含む etcd の定期的なメンテナンスを行うことを強く推奨しま

す。

6. システムには、状態遷移への対応として、指定された namespace 内のすべてのオブジェクトに対して反復処理する必要がある制御ループがいくつかあります。単一の namespace に特定タイプのオブジェクトの数が増えると、ループのコストが上昇し、特定の状態変更を処理する速度が低下します。この制限については、アプリケーションの各種要件を満たすのに十分な CPU、メモリー、およびディスクがシステムにあることが前提となっています。
7. 各サービスポートと各サービスのバックエンドには、**iptables** に対応するエントリーがありません。特定のサービスのバックエンドの数は、**Endpoints** オブジェクトのサイズに影響を与え、システム全体に送信されるデータのサイズに影響を与えます。
8. OpenShift Container Platform では、OpenShift Container Platform によってインストールされるカスタムリソース定義 (CRD)、OpenShift Container Platform と統合される製品、およびユーザーが作成した CRD を含むカスタムリソース定義 (CRD) の合計数が 512 に制限されません。作成された CRD の数が 512 を超える場合、**oc** コマンドリクエストのスロットリングが適用される可能性があります。

2.1.1. シナリオ例

例として、OpenShift Container Platform 4.13、OVN-Kubernetes ネットワークプラグイン、および以下のワークロードオブジェクトを使用して、500 個のワーカーノード (m5.2xl) がテストされ、サポートされています。

- デフォルトに加えて、200 個の namespace
- ノードあたり 60 Pod。30 台のサーバーと 30 台のクライアント Pod (合計 30k)
- 57 イメージストリーム/ns (合計 11.4k)
- サーバー Pod によってサポートされる 15 サービス/ns (合計 3k)
- 以前のサービスに裏打ちされた 15 ルート/ns (合計 3k)
- 20 シークレット/ns (合計 4k)
- 10 設定マップ/ns (合計 2k)
- 6 つのネットワークポリシー/ns (すべて拒否、インGRESSから許可、ネームスペース内ルールを含む)
- 57 ビルド/ns

次の要因は、クラスターのワークロードのスケールにプラスまたはマイナスの影響を与えることがわかっており、デプロイメントを計画するときにスケールの数値に考慮する必要があります。追加情報とガイダンスについては、営業担当者または [Red Hat サポート](#) にお問い合わせください。

- ノードあたりの Pod 数
- Pod あたりのコンテナ数
- 使用されるプローブのタイプ (liveness/readiness、exec/http など)
- ネットワークポリシーの数
- プロジェクトまたは namespace の数

- プロジェクトあたりのイメージストリーム数
- プロジェクトあたりのビルド数
- サービス/エンドポイントの数とタイプ
- ルート数
- シャード数
- シークレットの数
- config map の数
- API 呼び出しのレート、またはクラスターのチャーン。これは、クラスター設定内で物事が変化する速さの推定値です。
 - 5分間のウィンドウでの1秒あたりの Pod 作成リクエストの Prometheus クエリ: `sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))`
 - 5分間のウィンドウで1秒あたりのすべての API リクエストに対する Prometheus クエリ: `sum(irate(apiserver_request_count{}[5m]))`
- CPU のクラスターノードリソース消費量
- メモリーのクラスターノードリソース消費量

2.2. クラスターの最大値がテスト済みの OPENSIFT CONTAINER PLATFORM 環境および設定

2.2.1. AWS クラウドプラットフォーム:

ノード	フレーバー	vCPU	RAM(GiB)	ディスクタイプ	ディスクサイズ (GiB)/IOPS	カウント	リージョン
コントロールプレーン/etcd ^[1]	r5.4xlarge	16	128	gp3	220	3	us-west-2
インフラ ^[2]	m5.12xlarge	48	192	gp3	100	3	us-west-2
ワークロード ^[3]	m5.4xlarge	16	64	gp3	500 ^[4]	1	us-west-2
コンピューター	m5.2xlarge	8	32	gp3	100	3/25/250/500 ^[5]	us-west-2

1. etcd は遅延の影響を受けやすいため、ベースラインパフォーマンスが 3000 IOPS で毎秒 125 MiB の gp3 ディスクがコントロールプレーン/etcd ノードに使用されます。gp3 ボリュームはバーストパフォーマンスを使用しません。
2. インフラストラクチャーノードは、モニタリング、Ingress およびレジストリーコンポーネントをホストするために使用され、これにより、それらが大規模に実行する場合に必要なリソースを十分に確保することができます。
3. ワークロードノードは、パフォーマンスとスケーラビリティのワークロードジェネレーターを実行するための専用ノードです。
4. パフォーマンスおよびスケーラビリティのテストの実行中に収集される大容量のデータを保存するのに十分な領域を確保できるように、大きなディスクサイズが使用されます。
5. クラスタは反復的にスケーリングされ、パフォーマンスおよびスケーラビリティテストは指定されたノード数で実行されます。

2.2.2. IBM Power プラットフォーム

ノード	vCPU	RAM(GiB)	ディスクタイプ	ディスクサイズ (GiB)/IOPS	カウント
コントロールプレーン/etcd [1]	16	32	io1	GiB あたり 120/10 IOPS	3
インフラ [2]	16	64	gp2	120	2
ワークロード [3]	16	256	gp2	120 [4]	1
コンピューター	16	64	gp2	120	2 から 100 [5]

1. GiB あたり 120/10 IOPS の io1 ディスクがコントロールプレーン/etcd ノードに使用されます。
2. インフラストラクチャーノードは、モニタリング、Ingress およびレジストリーコンポーネントをホストするために使用され、これにより、それらが大規模に実行する場合に必要なリソースを十分に確保することができます。
3. ワークロードノードは、パフォーマンスとスケーラビリティのワークロードジェネレーターを実行するための専用ノードです。
4. パフォーマンスおよびスケーラビリティのテストの実行中に収集される大容量のデータを保存するのに十分な領域を確保できるように、大きなディスクサイズが使用されます。
5. クラスタは反復でスケーリングされます。

2.2.3. IBM Z プラットフォーム

ノード	vCPU [4]	RAM(GiB)[5]	ディスクタイプ	ディスクサイズ (GiB)/IOS	カウント
コントロールプレーン/etcd [1,2]	8	32	ds8k	300 / LCU 1	3
コンピューター [1,3]	8	32	ds8k	150 / LCU 2	4 ノード (ノードあたり 100/250/500 Pod にスケールリング)

1. ノードは2つの論理制御ユニット (LCU) 間で分散され、コントロールプレーン/etcd ノードのディスク I/O 負荷を最適化します。etcd の I/O 需要が他のワークロードに干渉してはなりません。
2. 100/250/500 Pod で同時に複数の反復を実行するテストには、4つのコンピューターノードが使用されます。まず、Pod をインスタンス化できるかどうかを評価するために、アイドル Pod が使用されました。次に、ネットワークと CPU を必要とするクライアント/サーバーのワークロードを使用して、ストレス下でのシステムの安定性を評価しました。クライアント Pod とサーバー Pod はペアで展開され、各ペアは2つのコンピューターノードに分散されました。
3. 個別のワークロードノードは使用されませんでした。ワークロードは、2つのコンピューターノード間のマイクロサービスワークロードをシミュレートします。
4. 使用されるプロセッサの物理的な数は、6つの Integrated Facilities for Linux (IFL) です。
5. 使用される物理メモリーの合計は 512 GiB です。

2.3. テスト済みのクラスタの最大値に基づく環境計画

重要

ノード上で物理リソースを過剰にサブスクリプションすると、Kubernetes スケジューラーが Pod の配置時に行うリソースの保証に影響が及びます。メモリスワップを防ぐために実行できる処置について確認してください。

一部のテスト済みの最大値については、単一の namespace/ユーザーが作成するオブジェクトでのみ変更されます。これらの制限はクラスタ上で数多くのオブジェクトが実行されている場合には異なります。

本書に記載されている数は、Red Hat のテスト方法、セットアップ、設定、およびチューニングに基づいています。これらの数は、独自のセットアップおよび環境に応じて異なります。

環境の計画時に、ノードに配置できる Pod 数を判別します。

$$\text{required pods per cluster} / \text{pods per node} = \text{total number of nodes needed}$$

ノードあたりの Pod のデフォルトの最大数は 250 です。ただし、ノードに適合する Pod 数はアプリ

ケーション自体によって異なります。「アプリケーション要件に合わせて環境計画を立てる方法」で説明されているように、アプリケーションのメモリー、CPU およびストレージの要件を検討してください。

シナリオ例

クラスターごとに 2200 の Pod のあるクラスターのスコープを設定する場合、ノードごとに最大 500 の Pod があることを前提として、最低でも 5 つのノードが必要になります。

$$2200 / 500 = 4.4$$

ノード数を 20 に増やす場合は、Pod 配分がノードごとに 110 の Pod に変わります。

$$2200 / 20 = 110$$

ここでは、以下ようになります。

$$\text{required pods per cluster} / \text{total number of nodes} = \text{expected pods per node}$$

OpenShift Container Platform には、SDN、DNS、Operator など、デフォルトですべてのワーカーノードで実行される複数のシステム Pod が付属しています。したがって、上記の式の結果は異なる場合があります。

2.4. アプリケーション要件に合わせて環境計画を立てる方法

アプリケーション環境の例を考えてみましょう。

Pod タイプ	Pod 数	最大メモリー	CPU コア数	永続ストレージ
apache	100	500 MB	0.5	1 GB
node.js	200	1 GB	1	1 GB
postgresql	100	1 GB	2	10 GB
JBoss EAP	100	1 GB	1	1 GB

推定要件: CPU コア 550 個、メモリー 450GB およびストレージ 1.4TB

ノードのインスタンスサイズは、希望に応じて増減を調整できます。ノードのリソースはオーバーコミットされることが多く、デプロイメントシナリオでは、小さいノードで数を増やしたり、大きいノードで数を減らしたりして、同じリソース量を提供することもできます。このデプロイメントシナリオでは、小さいノードで数を増やしたり、大きいノードで数を減らしたりして、同じリソース量を提供することもできます。運用上の敏捷性やインスタンスあたりのコストなどの要因を考慮する必要があります。

ノードのタイプ	数量	CPU	RAM (GB)
ノード (オプション 1)	100	4	16

ノードのタイプ	数量	CPU	RAM (GB)
ノード (オプション 2)	50	8	32
ノード (オプション 3)	25	16	64

アプリケーションによってはオーバーコミットの環境に適しているものもあれば、そうでないものもあります。たとえば、Java アプリケーションや Huge Page を使用するアプリケーションの多くは、オーバーコミットに対応できません。対象のメモリーは、他のアプリケーションに使用できません。上記の例では、環境は一般的な比率として約 30% オーバーコミットされています。

アプリケーション Pod は環境変数または DNS のいずれかを使用してサービスにアクセスできます。環境変数を使用する場合、それぞれのアクティブなサービスについて、変数が Pod がノードで実行される際に kubelet によって挿入されます。クラスター対応の DNS サーバーは、Kubernetes API で新規サービスの有無を監視し、それぞれに DNS レコードのセットを作成します。DNS がクラスター全体で有効にされている場合、すべての Pod は DNS 名でサービスを自動的に解決できるはずですが、DNS を使用したサービス検出は、5000 サービスを超える使用できる場合があります。サービス検出に環境変数を使用する場合、引数のリストは namespace で 5000 サービスを超える場合の許可される長さを超えると、Pod およびデプロイメントは失敗します。デプロイメントのサービス仕様ファイルのサービスリンクを無効にして、以下を解消します。

```
---
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: deployment-config-template
  creationTimestamp:
  annotations:
    description: This template will create a deploymentConfig with 1 replica, 4 env vars and a service.
    tags: ""
objects:
- apiVersion: apps.openshift.io/v1
  kind: DeploymentConfig
  metadata:
    name: deploymentconfig${IDENTIFIER}
  spec:
    template:
      metadata:
        labels:
          name: replicationcontroller${IDENTIFIER}
      spec:
        enableServiceLinks: false
        containers:
        - name: pause${IDENTIFIER}
          image: "${IMAGE}"
          ports:
          - containerPort: 8080
            protocol: TCP
          env:
          - name: ENVVAR1_${IDENTIFIER}
            value: "${ENV_VALUE}"
          - name: ENVVAR2_${IDENTIFIER}
            value: "${ENV_VALUE}"
          - name: ENVVAR3_${IDENTIFIER}
```

```

    value: "${ENV_VALUE}"
  - name: ENVVAR4_${IDENTIFIER}
    value: "${ENV_VALUE}"
  resources: {}
  imagePullPolicy: IfNotPresent
  capabilities: {}
  securityContext:
    capabilities: {}
    privileged: false
  restartPolicy: Always
  serviceAccount: ""
replicas: 1
selector:
  name: replicationcontroller${IDENTIFIER}
triggers:
- type: ConfigChange
strategy:
  type: Rolling
- apiVersion: v1
kind: Service
metadata:
  name: service${IDENTIFIER}
spec:
  selector:
    name: replicationcontroller${IDENTIFIER}
  ports:
  - name: serviceport${IDENTIFIER}
    protocol: TCP
    port: 80
    targetPort: 8080
  clusterIP: ""
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}
parameters:
- name: IDENTIFIER
  description: Number to append to the name of resources
  value: '1'
  required: true
- name: IMAGE
  description: Image to use for deploymentConfig
  value: gcr.io/google-containers/pause-amd64:3.0
  required: false
- name: ENV_VALUE
  description: Value to use for environment variables
  generate: expression
  from: "[A-Za-z0-9]{255}"
  required: false
labels:
  template: deployment-config-template

```

namespace で実行できるアプリケーション Pod の数は、環境変数がサービス検出に使用される場合にサービスの数およびサービス名の長さによって異なります。システムの **ARG_MAX** は、新規プロセスの引数の最大の長さを定義し、デフォルトで 2097152 バイト (2 MiB) に設定されます。Kubelet は、以下を含む namespace で実行するようにスケジューラされる各 Pod に環境変数を挿入します。

- `<SERVICE_NAME>_SERVICE_HOST=<IP>`
- `<SERVICE_NAME>_SERVICE_PORT=<PORT>`
- `<SERVICE_NAME>_PORT=tcp://<IP>:<PORT>`
- `<SERVICE_NAME>_PORT_<PORT>_TCP=tcp://<IP>:<PORT>`
- `<SERVICE_NAME>_PORT_<PORT>_TCP_PROTO=tcp`
- `<SERVICE_NAME>_PORT_<PORT>_TCP_PORT=<PORT>`
- `<SERVICE_NAME>_PORT_<PORT>_TCP_ADDR=<ADDR>`

引数の長さが許可される値を超え、サービス名の文字数がこれに影響する場合、namespace の Pod は起動に失敗し始めます。たとえば、5000 サービスを含む namespace では、サービス名の制限は 33 文字であり、これにより namespace で 5000 Pod を実行できます。

第3章 IBM Z & IBM(R) LINUXONE 環境で推奨されるホストの実践方法

このトピックでは、IBM Z および IBM® LinuxONE での OpenShift Container Platform のホストについての推奨プラクティスについて説明します。



注記

s390x アーキテクチャーは、多くの側面に固有のものです。したがって、ここで説明する推奨事項によっては、他のプラットフォームには適用されない可能性があります。



注記

特に指定がない限り、これらのプラクティスは IBM Z および IBM® LinuxONE での z/VM および Red Hat Enterprise Linux (RHEL) KVM インストールの両方に適用されます。

3.1. CPU のオーバーコミットの管理

高度に仮想化された IBM Z 環境では、インフラストラクチャーのセットアップとサイズ設定を慎重に計画する必要があります。仮想化の最も重要な機能の1つは、リソースのオーバーコミットを実行する機能であり、ハイパーバイザーレベルで実際に利用可能なリソースよりも多くのリソースを仮想マシンに割り当てます。これはワークロードに大きく依存し、すべてのセットアップに適用できる黄金律はありません。

設定によっては、CPU のオーバーコミットに関する以下のベストプラクティスを考慮してください。

- LPAR レベル (PR/SM ハイパーバイザー) で、利用可能な物理コア (IFL) を各 LPAR に割り当てないようにします。たとえば、4つの物理 IFL が利用可能な場合は、それぞれ4つの論理 IFL を持つ3つの LPAR を定義しないでください。
- LPAR 共有および重みを確認します。
- 仮想 CPU の数が多すぎると、パフォーマンスに悪影響を与える可能性があります。論理プロセッサが LPAR に定義されているよりも多くの仮想プロセッサをゲストに定義しないでください。
- ピーク時の負荷に対して、ゲストごとの仮想プロセッサ数を設定し、それ以上は設定しません。
- 小規模から始めて、ワークロードを監視します。必要に応じて、vCPU の数値を段階的に増やします。
- すべてのワークロードが、高いオーバーコミットメント率に適しているわけではありません。ワークロードが CPU 集約型である場合、パフォーマンスの問題なしに高い比率を実現できない可能性が高くなります。より多くの I/O 集約値であるワークロードは、オーバーコミットの使用率が高い場合でも、パフォーマンスの一貫性を保つことができます。

関連情報

- [z/VM Common Performance Problems and Solutions](#)
- [z/VM overcommitment considerations](#)
- [LPAR CPU management](#)

3.2. TRANSPARENT HUGE PAGES (THP) の無効

Transparent Huge Page (THP) は、Huge Page を作成し、管理し、使用するためのほとんどの要素を自動化しようとしています。THP は Huge Page を自動的に管理するため、すべてのタイプのワークロードに対して常に最適に処理される訳ではありません。THP は、多くのアプリケーションが独自の Huge Page を処理するため、パフォーマンス低下につながる可能性があります。したがって、THP を無効にすることを検討してください。

3.3. RECEIVE FLOW STEERING を使用したネットワークパフォーマンスの強化

Receive Flow Steering (RFS) は、ネットワークレイテンシーをさらに短縮して Receive Packet Steering (RPS) を拡張します。RFS は技術的には RPS をベースとしており、CPU キャッシュのヒットレートを増やして、パケット処理の効率を向上させます。RFS はこれを実現すると共に、計算に最も便利な CPU を決定することによってキューの長さを考慮し、キャッシュヒットが CPU 内で発生する可能性が高くなります。そのため、CPU キャッシュは無効化され、キャッシュを再構築するサイクルが少なくて済みます。これにより、パケット処理の実行時間を減らすのに役立ちます。

3.3.1. Machine Config Operator (MCO) を使用した RFS のアクティブ化

手順

1. 以下の MCO サンプルプロファイルを YAML ファイルにコピーします。たとえば、**enable-rfs.yaml** のようになります。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-enable-rfs
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=US-
ASCII,%23%20turn%20on%20Receive%20Flow%20Steering%20%28RFS%29%20for%20all
%20network%20interfaces%0ASUBSYSTEM%3D%3D%22net%22%2C%20ACTION%3D%
3D%22add%22%2C%20RUN%7Bprogram%7D%2B%3D%22/bin/bash%20-
c%20%27for%20x%20in%20/sys/%24DEVPATH/queues/rx-
%2A%3B%20do%20echo%208192%20%3E%20%24x/rps_flow_cnt%3B%20%20done%27
%22%0A
            filesystem: root
            mode: 0644
            path: /etc/udev/rules.d/70-persistent-net.rules
        - contents:
            source: data:text/plain;charset=US-
ASCII,%23%20define%20sock%20flow%20enbtried%20for%20%20Receive%20Flow%20Ste
ering%20%28RFS%29%0Anet.core.rps_sock_flow_entries%3D8192%0A
```

```
filesystem: root
mode: 0644
path: /etc/sysctl.d/95-enable-rfs.conf
```

2. MCO プロファイルを作成します。

```
$ oc create -f enable-rfs.yaml
```

3. **50-enable-rfs** という名前のエントリが表示されていることを確認します。

```
$ oc get mc
```

4. 非アクティブにするには、次のコマンドを実行します。

```
$ oc delete mc 50-enable-rfs
```

関連情報

- [OpenShift Container Platform on IBM Z: Tune your network performance with RFS](#)
- [RFS \(Receive Flow Steering\) の設定](#)
- [Scaling in the Linux Networking Stack](#)

3.4. ネットワーク設定の選択

ネットワークスタックは、OpenShift Container Platform などの Kubernetes ベースの製品の最も重要なコンポーネントの1つです。IBM Z 設定では、ネットワーク設定は選択したハイパーバイザーによって異なります。ワークロードとアプリケーションに応じて、最適なものは通常、ユースケースとトラフィックパターンによって異なります。

設定によっては、以下のベストプラクティスを考慮してください。

- トラフィックパターンを最適化するためにネットワークデバイスに関するすべてのオプションを検討してください。OSA-Express、RoCE Express、HiperSockets、z/VM VSwitch、Linux Bridge (KVM) の利点を調べて、セットアップに最大のメリットをもたらすオプションを決定します。
- 常に利用可能な最新の NIC バージョンを使用してください。たとえば、OSA Express 7S 10 GbE は、OSA Express 6S 10 GbE とトランザクションワークロードタイプと比べ、10 GbE アダプターよりも優れた改善を示しています。
- 各仮想スイッチは、追加のレイテンシーのレイヤーを追加します。
- ロードバランサーは、クラスター外のネットワーク通信に重要なロールを果たします。お使いのアプリケーションに重要な場合は、実稼働環境グレードのハードウェアロードバランサーの使用を検討してください。
- OpenShift Container Platform SDN では、ネットワークパフォーマンスに影響を与えるフローおよびルールが導入されました。コミュニケーションが重要なサービスの局所性から利益を得るには、Pod の親和性と配置を必ず検討してください。
- パフォーマンスと機能間のトレードオフのバランスを取ります。

関連情報

- [OpenShift Container Platform on IBM Z - Performance Experiences, Hints and Tips](#)
- [OpenShift Container Platform on IBM Z Networking Performance](#)
- [ノードのアフィニティールールを使用したノード上での Pod 配置の制御](#)

3.5. Z/VM の HYPERPAV でディスクのパフォーマンスが高いことを確認します。

DASD デバイスおよび ECKD デバイスは、IBM Z 環境で一般的に使用されているディスクタイプです。z/VM 環境で通常の OpenShift Container Platform 設定では、DASD ディスクがノードのローカルストレージをサポートするのに一般的に使用されます。HyperPAV エイリアスデバイスを設定して、z/VM ゲストをサポートする DASD ディスクに対してスループットおよび全体的な I/O パフォーマンスを向上できます。

ローカルストレージデバイスに HyperPAV を使用すると、パフォーマンスが大幅に向上します。ただし、スループットと CPU コストのトレードオフがあることに注意してください。

3.5.1. z/VM フルパックミニディスクを使用してノードで HyperPAV エイリアスをアクティブにするために Machine Config Operator (MCO) を使用します。

フルパックミニディスクを使用する z/VM ベースの OpenShift Container Platform セットアップの場合、すべてのノードで HyperPAV エイリアスをアクティベートして MCO プロファイルを利用できます。コントロールプレーンノードおよびコンピューターノードの YAML 設定を追加する必要があります。

手順

1. 以下の MCO サンプルプロファイルをコントロールプレーンノードの YAML ファイルにコピーします。たとえば、**05-master-kernelarg-hpav.yaml** です。

```
$ cat 05-master-kernelarg-hpav.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 05-master-kernelarg-hpav
spec:
  config:
    ignition:
      version: 3.1.0
    kernelArguments:
      - rd.dasd=800-805
```

2. 以下の MCO サンプルプロファイルをコンピューターノードの YAML ファイルにコピーします。たとえば、**05-worker-kernelarg-hpav.yaml** です。

```
$ cat 05-worker-kernelarg-hpav.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
```

```

name: 05-worker-kernelarg-hpav
spec:
  config:
    ignition:
      version: 3.1.0
    kernelArguments:
      - rd.dasd=800-805

```



注記

デバイス ID に合わせて **rd.dasd** 引数を変更する必要があります。

3. MCO プロファイルを作成します。

```
$ oc create -f 05-master-kernelarg-hpav.yaml
```

```
$ oc create -f 05-worker-kernelarg-hpav.yaml
```

4. 非アクティブにするには、次のコマンドを実行します。

```
$ oc delete -f 05-master-kernelarg-hpav.yaml
```

```
$ oc delete -f 05-worker-kernelarg-hpav.yaml
```

関連情報

- [Using HyperPAV for ECKD DASD](#)
- [Scaling HyperPAV alias devices on Linux guests on z/VM](#)

3.6. IBM Z ホストの RHEL KVM の推奨事項

KVM 仮想サーバーの環境を最適化すると、仮想サーバーと利用可能なリソースの可用性が大きく変わります。ある環境のパフォーマンスを向上させる同じアクションは、別の環境で悪影響を与える可能性があります。特定の設定に最適なバランスを見つけることは困難な場合があり、多くの場合は実験が必要です。

以下のセクションでは、IBM Z および IBM® LinuxONE 環境で RHEL KVM とともに OpenShift Container Platform を使用する場合のベストプラクティスについて説明します。

3.6.1. 仮想ブロックデバイスの I/O スレッドの使用

I/O スレッドを使用するように仮想ブロックデバイスを設定するには、仮想サーバー用に1つ以上の I/O スレッドを設定し、各仮想ブロックデバイスがこれらの I/O スレッドの1つを使用するように設定する必要があります。

以下の例は、`<iothreads>3</iothreads>` を指定し、3つの I/O スレッドを連続して1、2、および3に設定します。`iothread="2"` パラメーターは、ID 2 で I/O スレッドを使用するディスクデバイスのドライバー要素を指定します。

I/O スレッド仕様のサンプル

```

...
<domain>
  <iotreads>3</iotreads> ①
  ...
  <devices>
    ...
    <disk type="block" device="disk"> ②
  <driver ... iotread="2"/>
  </disk>
  ...
  </devices>
  ...
</domain>

```

① I/O スレッドの数。

② ディスクデバイスのドライバー要素。

スレッドは、ディスクデバイスの I/O 操作のパフォーマンスを向上させることができますが、メモリーおよび CPU リソースも使用します。同じスレッドを使用するように複数のデバイスを設定できます。スレッドからデバイスへの最適なマッピングは、利用可能なリソースとワークロードによって異なります。

少数の I/O スレッドから始めます。多くの場合は、すべてのディスクデバイスの単一の I/O スレッドで十分です。仮想 CPU の数を超えてスレッドを設定しないでください。アイドル状態のスレッドを設定しません。

virsh iotreadadd コマンドを使用して、特定のスレッド ID の I/O スレッドを稼働中の仮想サーバーに追加できます。

3.6.2. 仮想 SCSI デバイスの回避

SCSI 固有のインターフェイスを介してデバイスに対応する必要がある場合にのみ、仮想 SCSI デバイスを設定します。ホスト上でバックアップされるかどうかにかかわらず、仮想 SCSI デバイスではなく、ディスク領域を仮想ブロックデバイスとして設定します。

ただし、以下には、SCSI 固有のインターフェイスが必要になる場合があります。

- ホスト上で SCSI 接続のテープドライブ用の LUN。
- 仮想 DVD ドライブにマウントされるホストファイルシステムの DVD ISO ファイル。

3.6.3. ディスクについてのゲストキャッシュの設定

ホストではなく、ゲストでキャッシュするようにディスクデバイスを設定します。

ディスクデバイスのドライバー要素に **cache="none"** パラメーターおよび **io="native"** パラメーターが含まれていることを確認します。

```

<disk type="block" device="disk">
  <driver name="qemu" type="raw" cache="none" io="native" iotread="1"/>
  ...
</disk>

```

3.6.4. メモリーバルーンデバイスを除外します。

動的メモリーサイズが必要ない場合は、メモリーバルーンデバイスを定義せず、libvirt が管理者用に作成しないようにする必要があります。**memballoon** パラメーターを、ドメイン設定 XML ファイルの `devices` 要素の子として含めます。

- アクティブなプロファイルのリストを確認します。

```
<memballoon model="none"/>
```

3.6.5. ホストスケジューラーの CPU 移行アルゴリズムの調整



重要

影響を把握する専門家がない限り、スケジューラーの設定は変更しないでください。テストせずに実稼働システムに変更を適用せず、目的の効果を確認しないでください。

kernel.sched_migration_cost_ns パラメーターは、ナノ秒の間隔を指定します。タスクの最後の実行後、CPU キャッシュは、この間隔が期限切れになるまで有用なコンテンツを持つと見なされます。この間隔を大きくすると、タスクの移行が少なくなります。デフォルト値は 500000 ns です。

実行可能なプロセスがあるときに CPU アイドル時間が予想よりも長い場合は、この間隔を短くしてみてください。タスクが CPU またはノード間で頻繁にバウンスする場合は、それを増やしてみてください。

間隔を 60000 ns に動的に設定するには、以下のコマンドを入力します。

```
# sysctl kernel.sched_migration_cost_ns=60000
```

値を 60000 ns に永続的に変更するには、次のエントリーを **/etc/sysctl.conf** に追加します。

```
kernel.sched_migration_cost_ns=60000
```

3.6.6. cpuset cgroup コントローラーの無効化



注記

この設定は、cgroups バージョン 1 の KVM ホストにのみ適用されます。ホストで CPU ホットプラグを有効にするには、cgroup コントローラーを無効にします。

手順

1. 任意のエディターで **/etc/libvirt/qemu.conf** を開きます。
2. **cgroup_controllers** 行に移動します。
3. 行全体を複製し、コピーから先頭の番号記号 (#) を削除します。
4. **cpuset** エントリーを以下のように削除します。

```
cgroup_controllers = [ "cpu", "devices", "memory", "blkio", "cpuacct" ]
```

5. 新しい設定を有効にするには、libvirtd デーモンを再起動する必要があります。

- a. すべての仮想マシンを停止します。
- b. 以下のコマンドを実行します。

```
# systemctl restart libvirtd
```

- c. 仮想マシンを再起動します。

この設定は、ホストの再起動後も維持されます。

3.6.7. アイドル状態の仮想 CPU のポーリング期間の調整

仮想 CPU がアイドル状態になると、KVM は仮想 CPU のウェイクアップ条件をポーリングしてからホストリソースを割り当てます。ポーリングが `sysfs` の `/sys/module/kvm/parameters/halt_poll_ns` に配置される時間間隔を指定できます。指定された時間中、ポーリングにより、リソースの使用量を犠牲にして、仮想 CPU のウェイクアップレイテンシーが短縮されます。ワークロードに応じて、ポーリングの時間を長くしたり短くしたりすることが有益な場合があります。間隔はナノ秒で指定します。デフォルトは 50000 ns です。

- CPU の使用率が低い場合を最適化するには、小さい値または書き込み 0 を入力してポーリングを無効にします。

```
# echo 0 > /sys/module/kvm/parameters/halt_poll_ns
```

- トランザクションワークロードなどの低レイテンシーを最適化するには、大きな値を入力します。

```
# echo 80000 > /sys/module/kvm/parameters/halt_poll_ns
```

関連情報

- [Linux on IBM Z Performance Tuning for KVM](#)
- [IBM Z での仮想化の使用](#)

第4章 NODE TUNING OPERATOR の使用

Node Tuning Operator について説明し、この Operator を使用し、Tuned デーモンのオーケストレーションを実行してノードレベルのチューニングを管理する方法について説明します。

4.1. NODE TUNING OPERATOR について

Node Tuning Operator は、Tuned デーモンを調整することでノードレベルのチューニングを管理し、PerformanceProfile コントローラーを使用して低レイテンシーのパフォーマンスを実現するのに役立ちます。ほとんどの高パフォーマンスアプリケーションでは、一定レベルのカーネルのチューニングが必要です。Node Tuning Operator は、ノードレベルの `sysctl` の統一された管理インターフェイスをユーザーに提供し、ユーザーが指定するカスタムチューニングを追加できるよう柔軟性を提供します。

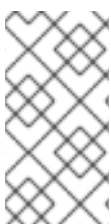
Operator は、コンテナ化された OpenShift Container Platform の Tuned デーモンを Kubernetes デーモンセットとして管理します。これにより、カスタムチューニング仕様が、デーモンが認識する形式でクラスターで実行されるすべてのコンテナ化された Tuned デーモンに渡されます。デーモンは、ノードごとに1つずつ、クラスターのすべてのノードで実行されます。

コンテナ化された Tuned デーモンによって適用されるノードレベルの設定は、プロファイルの変更をトリガーするイベントで、または終了シグナルの受信および処理によってコンテナ化された Tuned デーモンが正常に終了する際にロールバックされます。

Node Tuning Operator は、パフォーマンスプロファイルコントローラーを使用して自動チューニングを実装し、OpenShift Container Platform 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ます。

クラスター管理者は、以下のようなノードレベルの設定を定義するパフォーマンスプロファイルを設定します。

- カーネルを `kernel-rt` に更新します。
- ハウスキーピング用の CPU を選択します。
- 実行中のワークロード用の CPU を選択します。



注記

現在、CPU 負荷分散の無効化は `cgroup v2` ではサポートされていません。その結果、`cgroup v2` が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、`cgroup v2` を有効にすることは推奨されません。

Node Tuning Operator は、バージョン 4.1 以降における標準的な OpenShift Container Platform インストールの一部となっています。



注記

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

4.2. NODE TUNING OPERATOR 仕様サンプルへのアクセス

このプロセスを使用して Node Tuning Operator 仕様サンプルにアクセスします。

手順

- 次のコマンドを実行して、NodeTuningOperator 仕様の例にアクセスします。

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

デフォルトの CR は、OpenShift Container Platform プラットフォームの標準的なノードレベルのチューニングを提供することを目的としており、Operator 管理の状態を設定するためにのみ変更できます。デフォルト CR へのその他のカスタム変更は、Operator によって上書きされます。カスタムチューニングの場合は、独自のチューニングされた CR を作成します。新規に作成された CR は、ノード/Pod ラベルおよびプロファイルの優先順位に基づいて OpenShift Container Platform ノードに適用されるデフォルトの CR およびカスタムチューニングと組み合わせられます。



警告

特定の状況で Pod ラベルのサポートは必要なチューニングを自動的に配信する便利な方法ですが、この方法は推奨されず、とくに大規模なクラスターにおいて注意が必要です。デフォルトの調整された CR は Pod ラベル一致のない状態で提供されます。カスタムプロファイルが Pod ラベル一致のある状態で作成される場合、この機能はその時点で有効になります。Pod ラベル機能は、Node Tuning Operator の将来のバージョンで非推奨になる予定です。

4.3. クラスターに設定されるデフォルトのプロファイル

以下は、クラスターに設定されるデフォルトのプロファイルです。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=Optimize systems running OpenShift (provider specific parent profile)
        include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift
        name: openshift
      recommend:
    - profile: openshift-control-plane
      priority: 30
      match:
        - label: node-role.kubernetes.io/master
        - label: node-role.kubernetes.io/infra
    - profile: openshift-node
      priority: 40
```

OpenShift Container Platform 4.9 以降では、すべての OpenShift TuneD プロファイルが TuneD パッケージに含まれています。**oc exec** コマンドを使用して、これらのプロファイルの内容を表示できます。

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{-,control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

4.4. TUNED プロファイルが適用されていることの確認

クラスターノードに適用されている Tune D プロファイルを確認します。

```
$ oc get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

出力例

NAME	TUNED	APPLIED	DEGRADED	AGE
master-0	openshift-control-plane	True	False	6h33m
master-1	openshift-control-plane	True	False	6h33m
master-2	openshift-control-plane	True	False	6h33m
worker-a	openshift-node	True	False	6h28m
worker-b	openshift-node	True	False	6h28m

- **NAME:** Profile オブジェクトの名前。ノードごとに Profile オブジェクトが1つあり、それぞれの名前が一致します。
- **TUNED:** 適用する任意の TuneD プロファイルの名前。
- **APPLIED:** TuneD デーモンが任意のプロファイルを適用する場合は **True**。(true/False/Unknown)。
- **DEGRADED:** TuneD プロファイルのアプリケーション中にエラーが報告される場合は **True** (True/False/Unknown)
- **AGE:** Profile オブジェクトの作成からの経過時間。

ClusterOperator/node-tuning オブジェクトには、Operator とそのノードエージェントの状態に関する有用な情報も含まれています。たとえば、Operator の設定ミスは、**ClusterOperator/node-tuning** ステータスメッセージによって報告されます。

ClusterOperator/node-tuning オブジェクトに関するステータス情報を取得するには、次のコマンドを実行します。

```
$ oc get co/node-tuning -n openshift-cluster-node-tuning-operator
```

出力例

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE	MESSAGE
node-tuning	4.13.1	True	False	True	60m	1/5 Profiles with bootcmdline conflict

ClusterOperator/node-tuning またはプロファイルオブジェクトのステータスが **DEGRADED** の場合、追加情報が Operator またはオペランドログに提供されます。

4.5. カスタムチューニング仕様

Operator のカスタムリソース (CR) には 2 つの重要なセクションがあります。1 つ目のセクションの **profile:** は TuneD プロファイルおよびそれらの名前のリストです。2 つ目の **recommend:** は、プロファイル選択ロジックを定義します。

複数のカスタムチューニング仕様は、Operator の namespace に複数の CR として共存できます。新規 CR の存在または古い CR の削除は Operator によって検出されます。既存のカスタムチューニング仕様はすべてマージされ、コンテナ化された TuneD デーモンの適切なオブジェクトは更新されます。

管理状態

Operator 管理の状態は、デフォルトの Tuned CR を調整して設定されます。デフォルトで、Operator は Managed 状態であり、**spec.managementState** フィールドはデフォルトの Tuned CR に表示されません。Operator Management 状態の有効な値は以下のとおりです。

- Managed: Operator は設定リソースが更新されるとそのオペランドを更新します。
- Unmanaged: Operator は設定リソースへの変更を無視します。
- Removed: Operator は Operator がプロビジョニングしたオペランドおよびリソースを削除します。

プロファイルデータ

profile: セクションは、TuneD プロファイルおよびそれらの名前をリスト表示します。

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

推奨プロファイル

profile: 選択ロジックは、CR の **recommend:** セクションによって定義されます。**recommend:** セクションは、選択基準に基づくプロファイルの推奨項目のリストです。

```
recommend:
<recommend-item-1>
# ...
```

```
<recommend-item-n>
```

リストの個別項目:

```
- machineConfigLabels: ①
  <mcLabels> ②
  match: ③
  <match> ④
  priority: <priority> ⑤
  profile: <tuned_profile_name> ⑥
  operand: ⑦
  debug: <bool> ⑧
  tunedConfig:
    reapply_sysctl: <bool> ⑨
```

- ① オプション:
- ② キー/値の **MachineConfig** ラベルのディクショナリー。キーは一意である必要があります。
- ③ 省略する場合は、優先度の高いプロファイルが最初に一致するか、**machineConfigLabels** が設定されていない限り、プロファイルの一致が想定されます。
- ④ オプションのリスト。
- ⑤ プロファイルの順序付けの優先度。数値が小さいほど優先度が高くなります (**0** が最も高い優先度になります)。
- ⑥ 一致に適用する TuneD プロファイル。例: **tuned_profile_1**
- ⑦ オプションのオペランド設定。
- ⑧ TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は **true**、オフの場合は **false** です。デフォルトは **false** です。
- ⑨ TuneD デーモンの **reapply_sysctl** 機能をオンまたはオフにします。オプションは on で **true**、オフの場合は **false** です。

<match> は、以下のように再帰的に定義されるオプションの一覧です。

```
- label: <label_name> ①
  value: <label_value> ②
  type: <label_type> ③
  <match> ④
```

- ① ノードまたは Pod のラベル名。
- ② オプションのノードまたは Pod のラベルの値。省略されている場合も、**<label_name>** があるだけで一致条件を満たします。
- ③ オプションのオブジェクトタイプ (**node** または **pod**)。省略されている場合は、**node** が想定されます。
- ④ オプションの **<match>** リスト。

<match> が省略されない場合、ネストされたすべての <match> セクションが **true** に評価される必要もあります。そうでない場合には **false** が想定され、それぞれの <match> セクションのあるプロファイルは適用されず、推奨されません。そのため、ネスト化 (子の <match> セクション) は論理 AND 演算子として機能します。これとは逆に、<match> 一覧のいずれかの項目が一致する場合は、<match> の一覧全体が **true** に評価されます。そのため、リストは論理 OR 演算子として機能します。

machineConfigLabels が定義されている場合は、マシン設定プールベースのマッチングが指定の **recommend:** 一覧の項目に対してオンになります。<mcLabels> はマシン設定のラベルを指定します。マシン設定は、プロファイル <tuned_profile_name> についてカーネル起動パラメーターなどのホスト設定を適用するために自動的に作成されます。この場合は、マシン設定セレクターが <mcLabels> に一致するすべてのマシン設定プールを検索し、プロファイル <tuned_profile_name> を確認されるマシン設定プールが割り当てられるすべてのノードに設定する必要があります。マスターロールとワーカーのロールの両方を持つノードをターゲットにするには、マスターロールを使用する必要があります。

リスト項目の **match** および **machineConfigLabels** は論理 OR 演算子によって接続されます。match 項目は、最初にショートサーキット方式で評価されます。そのため、**true** と評価される場合、**machineConfigLabels** 項目は考慮されません。



重要

マシン設定プールベースのマッチングを使用する場合は、同じハードウェア設定を持つノードを同じマシン設定プールにグループ化することが推奨されます。この方法に従わない場合は、TuneD オペランドが同じマシン設定プールを共有する 2 つ以上のノードの競合するカーネルパラメーターを計算する可能性があります。

例: ノード/Pod ラベルベースのマッチング

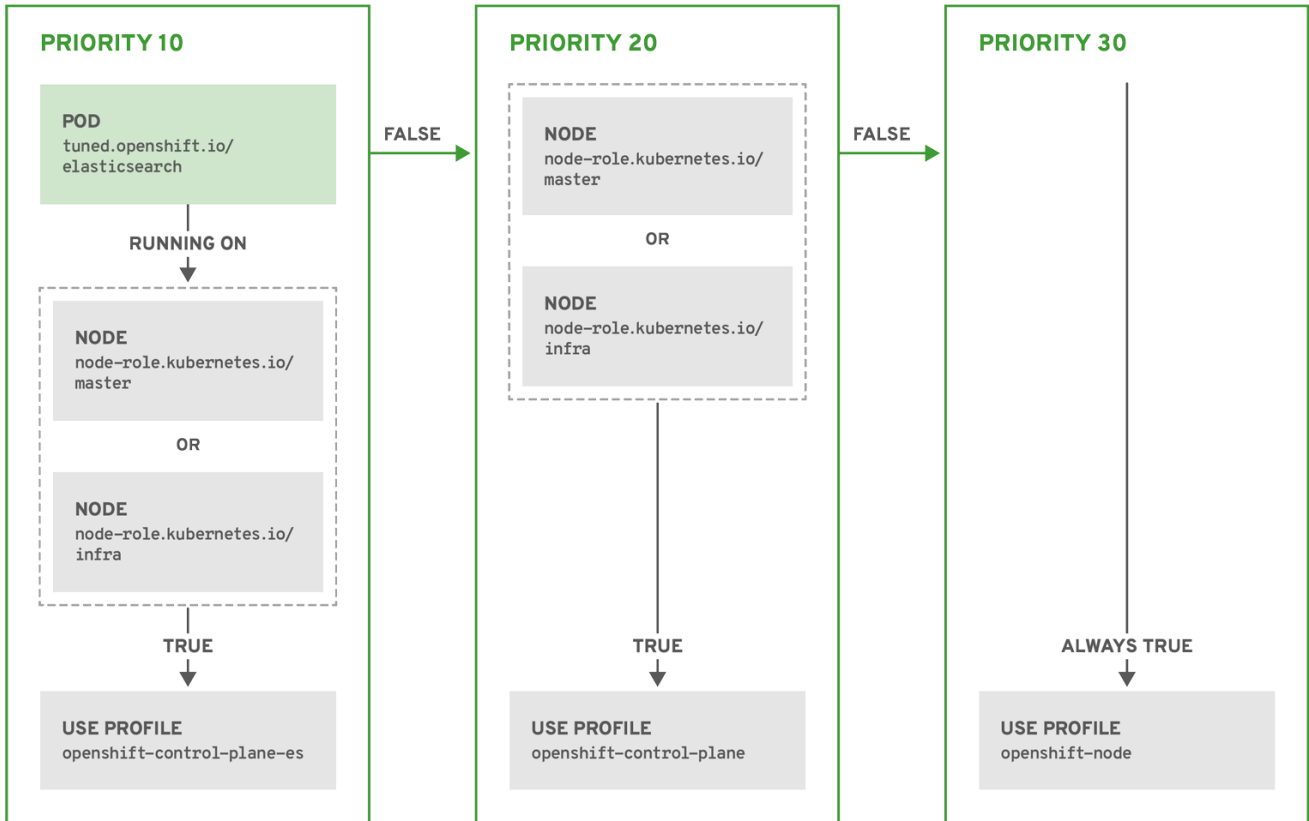
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

上記のコンテナ化された TuneD デーモンの CR は、プロファイルの優先順位に基づいてその **recommend.conf** ファイルに変換されます。最も高い優先順位 (10) を持つプロファイルは **openshift-control-plane-es** であるため、これが最初に考慮されます。指定されたノードで実行されるコンテナ化された TuneD デーモンは、同じノードに **tuned.openshift.io/elasticsearch** ラベルが設定された Pod が実行されているかどうかを確認します。これがない場合は、<match> セクション全体が **false** として評価されます。このラベルを持つこのような Pod がある場合に、<match> セクションが **true** に評価されるようにするには、ノードラベルを **node-role.kubernetes.io/master** または **node-role.kubernetes.io/infra** にする必要もあります。

優先順位が 10 のプロファイルのラベルが一致した場合は、**openshift-control-plane-es** プロファイルが適用され、その他のプロファイルは考慮されません。ノード/Pod ラベルの組み合わせが一致しない

場合は、2番目に高い優先順位プロファイル (**openshift-control-plane**) が考慮されます。このプロファイルは、コンテナ化された TuneD Pod が **node-role.kubernetes.io/master** または **node-role.kubernetes.io/infra** ラベルを持つノードで実行される場合に適用されます。

最後に、プロファイル **openshift-node** には最低の優先順位である **30** が設定されます。これには **<match>** セクションがないため、常に一致します。これは、より高い優先順位の他のプロファイルが指定されたノードで一致しない場合に **openshift-node** プロファイルを設定するために、最低の優先順位のノードが適用される汎用的な (catch-all) プロファイルとして機能します。



OPENSIFT_10_0319

例: マシン設定プールベースのマッチング

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
    [main]
    summary=Custom OpenShift node profile with an additional kernel parameter
    include=openshift-node
    [bootloader]
    cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

recommend:
  - machineConfigLabels:

```

```
machineconfiguration.openshift.io/role: "worker-custom"
priority: 20
profile: openshift-node-custom
```

ノードの再起動を最小限にするには、ターゲットノードにマシン設定プールのノードセクターが一致するラベルを使用してラベルを付け、上記の Tuned CR を作成してから、最後にカスタムのマシン設定プール自体を作成します。

クラウドプロバイダー固有の TuneD プロファイル

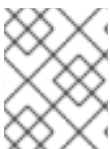
この機能により、すべてのクラウドプロバイダー固有のノードに、OpenShift Container Platform クラスタ上の特定のクラウドプロバイダーに合わせて特別に調整された TuneD プロファイルを簡単に割り当てることができます。これは、追加のノードラベルを追加したり、ノードをマシン設定プールにグループ化したりせずに実行できます。

この機能は、`<cloud-provider>://<cloud-provider-specific-id>` の形式で `spec.providerID` ノードオブジェクト値を利用して、NTO オペランドコンテナの `<cloud-provider>` の値で `/var/lib/tuned/provider` ファイルを書き込みます。その後、このファイルのコンテンツは TuneD により、プロバイダー `provider-<cloud-provider>` プロファイル (存在する場合) を読み込むために使用されます。

`openshift-control-plane` および `openshift-node` プロファイルの両方の設定を継承する `openshift` プロファイルは、条件付きプロファイルの読み込みを使用してこの機能を使用するよう更新されるようになりました。現時点で、NTO や TuneD にクラウドプロバイダー固有のプロファイルは含まれていません。ただし、すべてのクラウドプロバイダー固有のクラスターノードに適用されるカスタムプロファイル `provider-<cloud-provider>` を作成できます。

GCE クラウドプロバイダープロファイルの例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
    [main]
    summary=GCE Cloud provider-specific profile
    # Your tuning for GCE Cloud provider goes here.
  name: provider-gce
```



注記

プロファイルの継承により、`provider-<cloud-provider>` プロファイルで指定された設定は、`openshift` プロファイルとその子プロファイルによって上書きされます。

4.6. カスタムチューニングの例

デフォルト CR からの TuneD プロファイルの使用

以下の CR は、ラベル `tuned.openshift.io/ingress-node-label` を任意の値に設定した状態で OpenShift Container Platform ノードのカスタムノードレベルのチューニングを適用します。

例: `openshift-control-plane` TuneD プロファイルを使用したカスタムチューニング

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: ingress
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=A custom OpenShift ingress profile
      include=openshift-control-plane
      [sysctl]
      net.ipv4.ip_local_port_range="1024 65535"
      net.ipv4.tcp_tw_reuse=1
      name: openshift-ingress
  recommend:
    - match:
      - label: tuned.openshift.io/ingress-node-label
      priority: 10
      profile: openshift-ingress

```



重要

カスタムプロファイル作成者は、デフォルトの TuneD CR に含まれるデフォルトの調整されたデーモンプロファイルを組み込むことが強く推奨されます。上記の例では、デフォルトの **openshift-control-plane** プロファイルを使用してこれを実行します。

ビルトイン TuneD プロファイルの使用

NTO が管理するデーモンセットのロールアウトに成功すると、TuneD オペランドはすべて同じバージョンの TuneD デーモンを管理します。デーモンがサポートするビルトイン TuneD プロファイルをリスト表示するには、以下の方法で TuneD Pod をクエリーします。

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/ -name tuned.conf -printf '%h\n' | sed 's|^.*|'
```

このコマンドで取得したプロファイル名をカスタムのチューニング仕様で使用できます。

例: built-in hpc-compute TuneD プロファイルの使用

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-hpc-compute
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=Custom OpenShift node profile for HPC compute workloads
      include=openshift-node,hpc-compute
      name: openshift-node-hpc-compute
  recommend:

```

```
- match:
- label: tuned.openshift.io/openshift-node-hpc-compute
priority: 20
profile: openshift-node-hpc-compute
```

ビルトインの **hpc-compute** プロファイルに加えて、上記の例には、デフォルトの Tuned CR に同梱される **openshift-node** TuneD デーモンプロファイルが含まれており、コンピュータノードに OpenShift 固有のチューニングを使用します。

ホストレベルの sysctl のオーバーライド

`/run/sysctl.d/`、`/etc/sysctl.d/`、および `/etc/sysctl.conf` ホスト設定ファイルを使用して、実行時にさまざまなカーネルパラメーターを変更できます。OpenShift Container Platform は、実行時にカーネルパラメーターを設定する複数のホスト設定ファイルを追加します。たとえば、**net.ipv4-6**、**fs.inotify**、および **vm.max_map_count**。これらのランタイムパラメーターは、kubelet および Operator の開始前に、システムの基本的な機能調整を提供します。

reapply_sysctl オプションが **false** に設定されていない限り、Operator はこれらの設定をオーバーライドしません。このオプションを **false** に設定すると、**TuneD** はカスタムプロファイルを適用した後、ホスト設定ファイルからの設定を適用しません。

例: ホストレベルの sysctl のオーバーライド

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-no-reapply-sysctl
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
      [main]
      summary=Custom OpenShift profile
      include=openshift-node
      [sysctl]
      vm.max_map_count=>524288
      name: openshift-no-reapply-sysctl
  recommend:
    - match:
      - label: tuned.openshift.io/openshift-no-reapply-sysctl
      priority: 15
      profile: openshift-no-reapply-sysctl
      operand:
        tunedConfig:
          reapply_sysctl: false
```

4.7. サポートされている TUNED デーモンプラグイン

[main] セクションを除き、以下の TuneD プラグインは、Tuned CR の **profile:** セクションで定義されたカスタムプロファイルを使用する場合にサポートされます。

- audio
- cpu

- disk
- eeepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm
- bootloader

これらのプラグインの一部によって提供される動的チューニング機能の中に、サポートされていない機能があります。以下の TuneD プラグインは現時点でサポートされていません。

- script
- systemd



注記

TuneD ブートローダープラグインは、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみサポートします。

関連情報

- [利用可能な TuneD プラグイン](#)
- [TuneD を使い始める](#)

4.8. ホステッドクラスターにおけるノードのチューニング設定



重要

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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。

ホストされたクラスター内のノードでノードレベルのチューニングを設定するには、Node Tuning Operator を使用できます。ホストされたコントロールプレーンでは、**Tuned** オブジェクトを含む設定マップを作成し、ノードプールでそれらの設定マップを参照することで、ノードのチューニングを設定できます。

手順

1. チューニングされた有効なマニフェストを含む設定マップを作成し、ノードプールでマニフェストを参照します。次の例で **Tuned** マニフェストは、任意の値を持つ **tuned-1-node-label** ノードラベルを含むノード上で **vm.dirty_ratio** を 55 に設定するプロファイルを定義します。次の **ConfigMap** マニフェストを **tuned-1.yaml** という名前のファイルに保存します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-1
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: tuned-1
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
        - data: |
            [main]
            summary=Custom OpenShift profile
            include=openshift-node
            [sysctl]
            vm.dirty_ratio="55"
            name: tuned-1-profile
        recommend:
          - priority: 20
            profile: tuned-1-profile
```



注記

Tuned 仕様の **spec.recommend** セクションのエントリにラベルを追加しない場合は、ノードプールベースのマッチングが想定されるため、**spec.recommend** セクションの最も優先度の高いプロファイルがプール内のノードに適用されます。Tuned **.spec.recommend.match** セクションでラベル値を設定することにより、よりきめ細かいノードラベルベースのマッチングを実現できますが、ノードプールの **.spec.management.upgradeType** 値を **InPlace** に設定しない限り、ノードラベルはアップグレード中に保持されません。

2. 管理クラスターに **ConfigMap** オブジェクトを作成します。

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-1.yaml
```

3. ノードプールを編集するか作成して、ノードプールの **spec.tuningConfig** フィールドで **ConfigMap** オブジェクトを参照します。この例では、2つのノードを含む **nodepool-1** という名前の **NodePool** が1つだけあることを前提としています。

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  ...
  name: nodepool-1
  namespace: clusters
  ...
spec:
  ...
  tuningConfig:
    - name: tuned-1
status:
  ...
```



注記

複数のノードプールで同じ設定マップを参照できます。ホストされたコントロールプレーンでは、Node Tuning Operator はノードプール名と namespace のハッシュを Tuned CR の名前に追加してそれらを区別します。このケース以外では、同じホストクラスターの異なる Tuned CR に同じ名前の複数の Tuned プロファイルを作成しないでください。

検証

これで **Tuned** マニフェストを含む **ConfigMap** オブジェクトを作成し、それを **NodePool** で参照しました。次に、Node Tuning Operator は **Tuned** オブジェクトをホストされたクラスターに同期します。どの **Tuned** オブジェクトが定義されているか、どの Tuned プロファイルが各ノードに適用されているかを確認できます。

1. ホストされたクラスター内の **Tuned** オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

出力例

```
NAME    AGE
default 7m36s
rendered 7m36s
tuned-1 65s
```

2. ホストされたクラスター内の **Profile** オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

出力例

```
NAME                TUNED          APPLIED  DEGRADED  AGE
nodepool-1-worker-1  tuned-1-profile True     False     7m43s
nodepool-1-worker-2  tuned-1-profile True     False     7m14s
```



注記

カスタムプロファイルが作成されていない場合は、**openshift-node** プロファイルがデフォルトで適用されます。

3. チューニングが正しく適用されたことを確認するには、ノードでデバッグシェルを開始し、`sysctl` 値を確認します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host sysctl vm.dirty_ratio
```

出力例

```
vm.dirty_ratio = 55
```

4.9. カーネルブートパラメーターを設定することによる、ホストされたクラスタの高度なノードチューニング



重要

ホストされたコントロールプレーンは、テクノロジープレビュー機能としてのみ利用できます。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。

カーネルブートパラメーターの設定が必要な、ホストされたコントロールプレーンでのより高度なチューニングについては、Node Tuning Operator を使用することもできます。次の例は、Huge Page が予約されたノードプールを作成する方法を示しています。

手順

1. サイズが 2 MB の 10 個の Huge Page を作成するための **Tuned** オブジェクトマニフェストを含む **ConfigMap** オブジェクトを作成します。この **ConfigMap** マニフェストを **tuned-hugepages.yaml** という名前のファイルに保存します。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: tuned-hugepages
  namespace: clusters
data:
  tuning: |
    apiVersion: tuned.openshift.io/v1
    kind: Tuned
    metadata:
      name: hugepages
      namespace: openshift-cluster-node-tuning-operator
    spec:
      profile:
      - data: |
          [main]
          summary=Boot time configuration for hugepages
          include=openshift-node
          [bootloader]
          cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50
          name: openshift-node-hugepages
      recommend:
      - priority: 20
        profile: openshift-node-hugepages

```



注記

.spec.recommend.match フィールドは意図的に空白のままにしています。この場合、この **Tuned** オブジェクトは、この **ConfigMap** オブジェクトが参照されているノードプール内のすべてのノードに適用されます。同じハードウェア設定を持つノードを同じノードプールにグループ化します。そうしないと、TuneD オペランドは、同じノードプールを共有する 2 つ以上のノードに対して競合するカーネルパラメーターを計算する可能性があります。

2. 管理クラスターに **ConfigMap** オブジェクトを作成します。

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-hugepages.yaml
```

3. **NodePool** マニフェスト YAML ファイルを作成し、**NodePool** のアップグレードタイプをカスタマイズして、**spec.tuningConfig** セクションで作成した **ConfigMap** オブジェクトを参照します。**hypershift** CLI を使用して **NodePool** マニフェストを作成し、**hugepages-nodepool.yaml** という名前のファイルに保存します。

```

NODEPOOL_NAME=hugepages-example
INSTANCE_TYPE=m5.2xlarge
NODEPOOL_REPLICAS=2

```

```
hypershift create nodepool aws \
```

```

--cluster-name $CLUSTER_NAME \
--name $NODEPOOL_NAME \
--node-count $NODEPOOL_REPLICAS \
--instance-type $INSTANCE_TYPE \
--render > hugepages-nodepool.yaml

```

4. `hugepages-nodepool.yaml` ファイルで、`.spec.management.upgradeType` を `InPlace` に設定し、作成した `tuned-hugepages ConfigMap` オブジェクトを参照するように `.spec.tuningConfig` を設定します。

```

apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
  name: hugepages-nodepool
  namespace: clusters
...
spec:
  management:
    ...
    upgradeType: InPlace
    ...
  tuningConfig:
    - name: tuned-hugepages

```



注記

新しい `MachineConfig` オブジェクトを適用するときに不要なノードの再作成を回避するには、`.spec.management.upgradeType` を `InPlace` に設定します。`Replace` アップグレードタイプを使用する場合、ノードは完全に削除され、TuneD オペランドが計算した新しいカーネルブートパラメーターを適用すると、新しいノードでノードを置き換えることができます。

5. 管理クラスターに `NodePool` を作成します。

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f hugepages-nodepool.yaml
```

検証

ノードが使用可能になると、コンテナ化された TuneD デーモンが、適用された Tuned プロファイルに基づいて、必要なカーネルブートパラメーターを計算します。ノードの準備が整い、一度再起動して生成された `MachineConfig` オブジェクトを適用したら、TuneD プロファイルが適用され、カーネルブートパラメーターが設定されていることを確認できます。

1. ホストされたクラスター内の `Tuned` オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

出力例

```

NAME          AGE
default       123m
hugepages-8dfb1fed 1m23s
rendered      123m

```

2. ホストされたクラスター内の **Profile** オブジェクトを一覧表示します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

出力例

NAME	TUNED	APPLIED	DEGRADED	AGE
nodepool-1-worker-1	openshift-node	True	False	132m
nodepool-1-worker-2	openshift-node	True	False	131m
hugepages-nodepool-worker-1	openshift-node-hugepages	True	False	4m8s
hugepages-nodepool-worker-2	openshift-node-hugepages	True	False	3m57s

新しい **NodePool** の両方のワーカーノードには、**openshift-node-hugepages** プロファイルが適用されています。

3. チューニングが正しく適用されたことを確認するには、ノードでデバッグシェルを起動し、**/proc/cmdline** を確認します。

```
$ oc --kubeconfig="$HC_KUBECONFIG" debug node/nodepool-1-worker-1 -- chroot /host cat /proc/cmdline
```

出力例

```
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-... hugepagesz=2M hugepages=50
```

関連情報

ホストされたコントロールプレーンの詳細は、[ホストされたコントロールプレーン \(テクノロジープレビュー\)](#) を参照してください。

第5章 CPU マネージャーおよび TOPOLOGY MANAGER の使用

CPU マネージャーは、CPU グループを管理して、ワークロードを特定の CPU に制限します。

CPU マネージャーは、以下のような属性が含まれるワークロードに有用です。

- できるだけ長い CPU 時間が必要な場合
- プロセッサのキャッシュミスの影響を受ける場合
- レイテンシーが低いネットワークアプリケーションの場合
- 他のプロセスと連携し、単一のプロセッサキャッシュを共有することに利点がある場合

Topology Manager は、CPU マネージャー、デバイスマネージャー、およびその他の Hint Provider からヒントを収集し、同じ Non-Uniform Memory Access (NUMA) ノード上のすべての QoS (Quality of Service) クラスについて CPU、SR-IOV VF、その他デバイスリソースなどの Pod リソースを調整します。

Topology Manager は、収集したヒントのトポロジー情報を使用し、設定される Topology Manager ポリシーおよび要求される Pod リソースに基づいて、pod がノードから許可されるか、拒否されるかどうかを判別します。

Topology Manager は、ハードウェアアクセラレーターを使用して低遅延 (latency-critical) の実行と高スループットの並列計算をサポートするワークロードの場合に役立ちます。

Topology Manager を使用するには、**static** ポリシーで CPU マネージャーを設定する必要があります。

5.1. CPU マネージャーの設定

手順

1. オプション: ノードにラベルを指定します。

```
# oc label node perf-node.example.com cpumanager=true
```

2. CPU マネージャーを有効にする必要のあるノードの **MachineConfigPool** を編集します。この例では、すべてのワーカーで CPU マネージャーが有効にされています。

```
# oc edit machineconfigpool worker
```

3. ラベルをワーカーのマシン設定プールに追加します。

```
metadata:
  creationTimestamp: 2020-xx-xxx
  generation: 3
  labels:
    custom-kubelet: cpumanager-enabled
```

4. **KubeletConfig**、**cpumanager-kubeletconfig.yaml**、カスタムリソース (CR) を作成します。直前の手順で作成したラベルを参照し、適切なノードを新規の kubelet 設定で更新します。**machineConfigPoolSelector** セクションを参照してください。

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

```
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static ❶
    cpuManagerReconcilePeriod: 5s ❷
```

❶ ポリシーを指定します。

- **none**このポリシーは、既存のデフォルト CPU アフィニティースキームを明示的に有効にし、スケジューラーが自動的に実行するもの以外のアフィニティを提供しません。これはデフォルトポリシーになります。
- **static**このポリシーは、整数の CPU 要求を持つ保証された Pod 内のコンテナを許可します。また、ノードの排他的 CPU へのアクセスも制限します。**static** の場合は、小文字の **s** を使用する必要があります。

❷ オプション: CPU マネージャーの調整頻度を指定します。デフォルトは **5s** です。

5. 動的な kubelet 設定を作成します。

```
# oc create -f cpumanager-kubeletconfig.yaml
```

これにより、CPU マネージャー機能が kubelet 設定に追加され、必要な場合には Machine Config Operator (MCO) がノードを再起動します。CPU マネージャーを有効にするために再起動する必要はありません。

6. マージされた kubelet 設定を確認します。

```
# oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep ownerReference -A7
```

出力例

```
"ownerReferences": [
  {
    "apiVersion": "machineconfiguration.openshift.io/v1",
    "kind": "KubeletConfig",
    "name": "cpumanager-enabled",
    "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
  }
]
```

7. ワーカーで更新された **kubelet.conf** を確認します。

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

出力例


```
cpuManagerPolicy: static ①
cpuManagerReconcilePeriod: 5s ②
```

- ① **cpuManagerPolicy** は、**KubeletConfig** CR の作成時に定義されます。
- ② **cpuManagerReconcilePeriod** は、**KubeletConfig** CR の作成時に定義されます。

8. コア1つまたは複数を用意する Pod を作成します。制限および要求の CPU の値は整数にする必要があります。これは、対象の Pod 専用のコア数です。

```
# cat cpumanager-pod.yaml
```

出力例

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
  nodeSelector:
    cpumanager: "true"
```

9. Pod を作成します。

```
# oc create -f cpumanager-pod.yaml
```

10. Pod がラベル指定されたノードにスケジューリングされていることを確認します。

```
# oc describe pod cpumanager
```

出力例

```
Name:          cpumanager-6cqz7
Namespace:     default
Priority:       0
PriorityClassName: <none>
Node: perf-node.example.com/xxx.xx.xx.xxx
...
Limits:
  cpu: 1
  memory: 1G
Requests:
```

```

cpu:    1
memory: 1G
...
QoS Class:    Guaranteed
Node-Selectors: cpumanager=true

```

11. **cgroups** が正しく設定されていることを確認します。 **pause** プロセスのプロセス ID (PID) を取得します。

```

# |—init.scope
| |—1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
| |—kubepods.slice
| |   |—kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
| |   |   |—crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
| |   |   |—32706 /pause
| |
|

```

QoS (quality of service) 層 **Guaranteed** の Pod は、 **kubepods.slice** に配置されます。他の QoS 層の Pod は、 **kubepods** の子である **cgroups** に配置されます。

```

# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done

```

出力例

```

cpuset.cpus 1
tasks 32706

```

12. 対象のタスクで許可される CPU リストを確認します。

```

# grep ^Cpus_allowed_list /proc/32706/status

```

出力例

```

Cpus_allowed_list: 1

```

13. システム上の別の Pod (この場合は **burstable** QoS 層にある Pod) が、 **Guaranteed** Pod に割り当てられたコアで実行できないことを確認します。

```

# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus

0
# oc describe node perf-node.example.com

```

出力例

```

...
Capacity:
attachable-volumes-aws-ebs: 39
cpu:                          2

```

```

ephemeral-storage:      124768236Ki
hugepages-1Gi:          0
hugepages-2Mi:          0
memory:                 8162900Ki
pods:                   250
Allocatable:
attachable-volumes-aws-ebs: 39
cpu:                    1500m
ephemeral-storage:      124768236Ki
hugepages-1Gi:          0
hugepages-2Mi:          0
memory:                 7548500Ki
pods:                   250
-----
-
default                 cpumanager-6cqz7      1 (66%)   1 (66%)   1G (12%)
1G (12%)                29m
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource                Requests      Limits
-----
cpu                     1440m (96%)   1 (66%)

```

この仮想マシンには、2つのCPUコアがあります。**system-reserved**設定は500ミリコアを予約し、**Node Allocatable**の量になるようにノードの全容量からコアの半分を引きます。ここで**Allocatable CPU**は1500ミリコアであることを確認できます。これは、それぞれがコアを1つ受け入れるので、CPU マネージャー Pod の1つを実行できることを意味します。1つのコア全体は1000ミリコアに相当します。2つ目のPodをスケジュールしようとする場合、システムはPodを受け入れますが、これがスケジュールされることはありません。

```

NAME                READY STATUS  RESTARTS  AGE
cpumanager-6cqz7    1/1   Running  0         33m
cpumanager-7qc2t    0/1   Pending  0         11s

```

5.2. TOPOLOGY MANAGER ポリシー

Topology Manager は、CPU マネージャーや Device Manager などの Hint Provider からトポロジーのヒントを収集し、収集したヒントを使用して **Pod** リソースを調整することで、すべての QoS (Quality of Service) クラスの **Pod** リソースを調整します。

Topology Manager は、**cpumanager-enabled** という名前の **KubeletConfig** カスタムリソース (CR) で割り当てる 4 つの割り当てポリシーをサポートしています。

none ポリシー

これはデフォルトのポリシーで、トポロジーの配置は実行しません。

best-effort ポリシー

best-effort トポロジー管理ポリシーを持つ Pod のそれぞれのコンテナの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は、そのコンテナの推奨される NUMA ノードのアフィニティを保存します。アフィニティが優先されない場合、Topology Manager はこれを保管し、ノードに対して Pod を許可しません。

restricted ポリシー

restricted トポロジー管理ポリシーを持つ Pod のそれぞれのコンテナの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は、そのコンテナの推奨される NUMA ノードのアフィニティを保存します。アフィニティが優先されない場合、Topology Manager はこの Pod をノードから拒否します。これにより、Pod が Pod の受付の失敗により **Terminated** 状態になります。

single-numa-node ポリシー

single-numa-node トポロジー管理ポリシーがある Pod のそれぞれのコンテナの場合、kubelet は各 Hint Provider を呼び出してそれらのリソースの可用性を検出します。この情報を使用して、Topology Manager は単一の NUMA ノードのアフィニティが可能かどうかを判別します。可能である場合、Pod はノードに許可されます。単一の NUMA ノードアフィニティが使用できない場合には、Topology Manager は Pod をノードから拒否します。これにより、Pod は Pod の受付失敗と共に Terminated (終了) 状態になります。

5.3. TOPOLOGY MANAGER のセットアップ

Topology Manager を使用するには、**cpumanager-enabled** という名前の **KubeletConfig** カスタムリソース (CR) で割り当てポリシーを設定する必要があります。CPU マネージャーをセットアップしている場合は、このファイルが存在している可能性があります。ファイルが存在しない場合は、作成できません。

前提条件

- CPU マネージャーのポリシーを **static** に設定します。

手順

Topology Manager をアクティブにするには、以下を実行します。

1. カスタムリソースで Topology Manager 割り当てポリシーを設定します。

```
$ oc edit KubeletConfig cpumanager-enabled

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static ①
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node ②
```

- ① このパラメーターは、小文字の **s** で **static** にする必要があります。
- ② 選択した Topology Manager 割り当てポリシーを指定します。このポリシーは **single-numa-node** になります。使用できる値は、**default**、**best-effort**、**restricted**、**single-numa-node** です。

5.4. POD の TOPOLOGY MANAGER ポリシーとの対話

以下のサンプル **Pod** 仕様は、Pod の Topology Manger との対話について説明しています。

以下の Pod は、リソース要求や制限が指定されていないために **BestEffort** QoS クラスで実行されません。

```
spec:
  containers:
  - name: nginx
    image: nginx
```

以下の Pod は、要求が制限よりも小さいために **Burstable** QoS クラスで実行されます。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

選択したポリシーが **none** 以外の場合は、Topology Manager はこれらの **Pod** 仕様のいずれかも考慮しません。

以下の最後のサンプル Pod は、要求が制限と等しいために **Guaranteed** QoS クラスで実行されます。

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

Topology Manager はこの Pod を考慮します。Topology Manager はヒントプロバイダー (CPU マネージャーおよび Device Manager) を参照して、Pod のトポロジーヒントを取得します。

Topology Manager はこの情報を使用して、このコンテナに最適なトポロジーを保管します。この Pod の場合、CPU マネージャーおよびデバイスマネージャーは、リソース割り当ての段階でこの保存された情報を使用します。

第6章 NUMA 対応ワークロードのスケジューリング

NUMA 対応のスケジューリングと、それを使用して OpenShift Container Platform クラスタに高パフォーマンスのワークロードをデプロイする方法について学びます。

NUMA Resources Operator を使用すると、同じ NUMA ゾーンで高パフォーマンスのワークロードをスケジュールすることができます。これは、利用可能なクラスタードの NUMA リソースを報告するノードリソースエクスポートエージェントと、ワークロードを管理するセカンダリースケジューラーをデプロイします。

6.1. NUMA 対応のスケジューリングについて

Non-Uniform Memory Access (NUMA) は、異なる CPU が異なるメモリー領域に異なる速度でアクセスできるようにするコンピュートプラットフォームアーキテクチャーです。NUMA リソーストポロジーは、コンピュートノード内の相互に関連する CPU、メモリー、および PCI デバイスの位置を指しています。共同配置されたリソースは、同じ NUMA ゾーンにあるとされています。高性能アプリケーションの場合、クラスタは単一の NUMA ゾーンで Pod ワークロードを処理する必要があります。

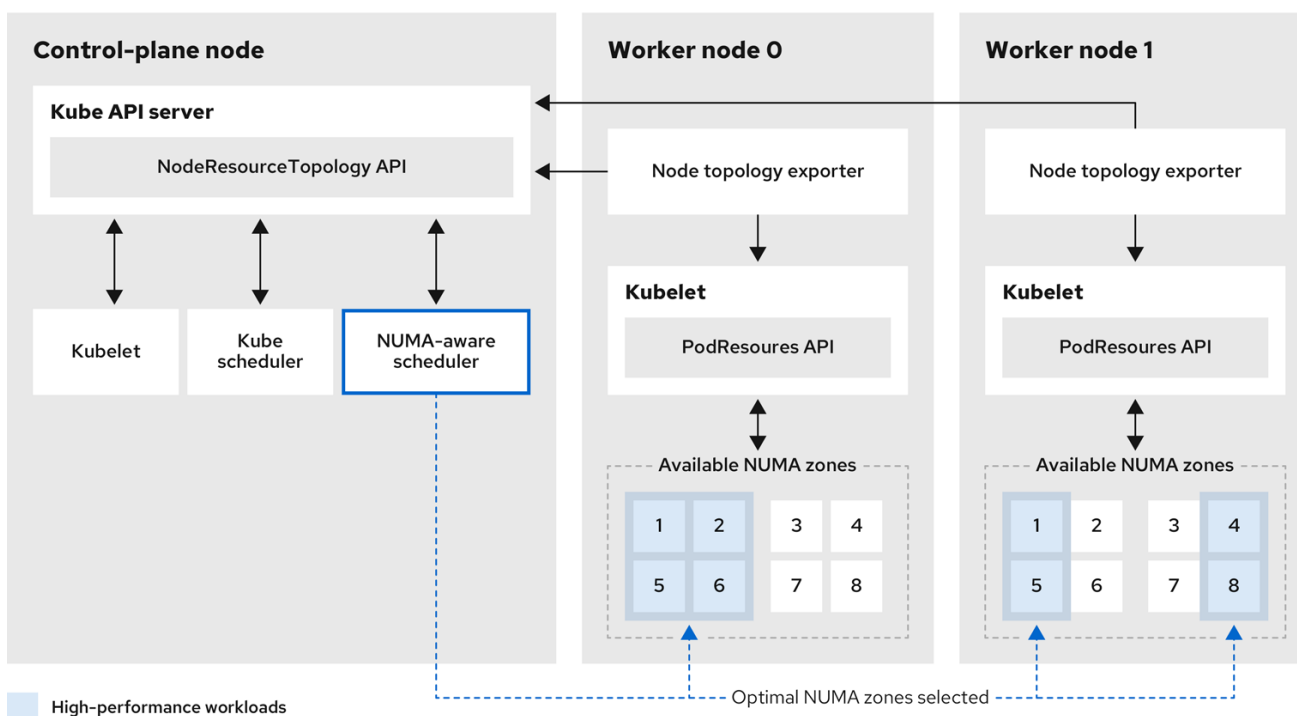
NUMA アーキテクチャーにより、複数のメモリーコントローラーを備えた CPU は、メモリーが配置されている場所に関係なく、CPU コンプレックス全体で使用可能なメモリーを使用できます。これにより、パフォーマンスを犠牲にして柔軟性を高めることができます。NUMA ゾーン外のメモリーを使用してワークロードを処理する CPU は、単一の NUMA ゾーンで処理されるワークロードよりも遅くなります。また、I/O に制約のあるワークロードの場合、離れた NUMA ゾーンのネットワークインターフェイスにより、情報がアプリケーションに到達する速度が低下します。通信ワークロードなどの高性能ワークロードは、これらの条件下では仕様どおりに動作できません。NUMA 対応のスケジューリングは、要求されたクラスタコンピュートリソース (CPU、メモリー、デバイス) を同じ NUMA ゾーンに配置して、レイテンシーの影響を受けやすいワークロードや高性能なワークロードを効率的に処理します。また、NUMA 対応のスケジューリングにより、コンピュートノードあたりの Pod 密度を向上させ、リソース効率を高めています。

Node Tuning Operator のパフォーマンスプロファイルを NUMA 対応スケジューリングと統合することで、CPU アフィニティをさらに設定し、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を最適化できます。

デフォルトの OpenShift Container Platform Pod スケジューラーのスケジューリングロジックは、個々の NUMA ゾーンではなく、コンピュートノード全体の利用可能なリソースを考慮します。kubelet トポロジーマネージャーで最も制限的なリソースアライメントが要求された場合、Pod をノードに許可するときにエラー状態が発生する可能性があります。逆に、最も制限的なリソース調整が要求されていない場合、Pod は適切なリソース調整なしでノードに許可され、パフォーマンスが低下したり予測不能になったりする可能性があります。たとえば、Pod スケジューラーが Pod の要求されたリソースが利用可能かどうかわからないために、Pod スケジューラーが保証された Pod ワークロードに対して次善のスケジューリング決定を行うと、**Topology Affinity Error** ステータスを伴う Pod 作成の暴走が発生する可能性があります。スケジュールの不一致の決定により、Pod の起動が無期限に遅延する可能性があります。また、クラスタの状態とリソースの割り当てによっては、Pod のスケジューリングの決定が適切でないと、起動の試行が失敗するためにクラスタに余分な負荷がかかる可能性があります。

NUMA Resources Operator は、カスタム NUMA リソースのセカンダリースケジューラーおよびその他のリソースをデプロイして、デフォルトの OpenShift Container Platform Pod スケジューラーの欠点を軽減します。次の図は、NUMA 対応 Pod スケジューリングの俯瞰的な概要を示しています。

図6.1 NUMA 対応スケジューリングの概要



216_OpenShift_0222

NodeResourceTopology API

NodeResourceTopology API は、各コンピュータードで使用可能な NUMA ゾーンリソースを記述します。

NUMA 対応スケジューラー

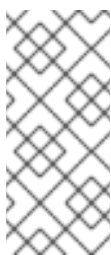
NUMA 対応のセカンダリースケジューラーは、利用可能な NUMA ゾーンに関する情報を **NodeResourceTopology** API から受け取り、最適に処理できるノードで高パフォーマンスのワークロードをスケジューリングします。

ノードトポロジーエクスポーター

ノードトポロジーエクスポーターは、各コンピュータードで使用可能な NUMA ゾーンリソースを **NodeResourceTopology** API に公開します。ノードトポロジーエクスポーターデーモンは、**PodResources** API を使用して、kubelet からのリソース割り当てを追跡します。

PodResources API

PodResources API は各ノードに対してローカルであり、リソーストポロジーと利用可能なリソースを kubelet に公開します。



注記

PodResources API の **List** エンドポイントは、特定のコンテナに割り当てられた排他的な CPU を公開します。API は、共有プールに属する CPU は公開しません。

GetAllocatableResources エンドポイントは、ノード上で使用できる割り当て可能なリソースを公開します。

関連情報

- クラスターでセカンダリー Pod スケジューラーを実行する方法と、セカンダリー Pod スケジューラーを使用して Pod をデプロイする方法の詳細については、[セカンダリースケジューラーを使用した Pod のスケジューリング](#) を参照してください。

6.2. NUMA RESOURCES OPERATOR のインストール

NUMA Resources Operator は、NUMA 対応のワークロードとデプロイメントをスケジュールできるリソースをデプロイします。OpenShift Container Platform CLI または Web コンソールを使用して NUMA Resources Operator をインストールできます。

6.2.1. CLI を使用した NUMA Resources Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

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

手順

1. NUMA Resources Operator の namespace を作成します。
 - a. 以下の YAML を **nro-namespace.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-numaresources
```

- b. 以下のコマンドを実行して **Namespace** CR を作成します。

```
$ oc create -f nro-namespace.yaml
```

2. NUMA Resources Operator の Operator グループを作成します。
 - a. 以下の YAML を **nro-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: numaresources-operator
  namespace: openshift-numaresources
spec:
  targetNamespaces:
    - openshift-numaresources
```

- b. 以下のコマンドを実行して **OperatorGroup** CR を作成します。

```
$ oc create -f nro-operatorgroup.yaml
```

3. NUMA Resources Operator のサブスクリプションを作成します。

- a. 以下の YAML を **nro-sub.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: numaresources-operator
  namespace: openshift-numaresources
spec:
  channel: "4.13"
  name: numaresources-operator
  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 nro-sub.yaml
```

検証

1. **openshift-numaresources** namespace の 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。以下のコマンドを実行します。

```
$ oc get csv -n openshift-numaresources
```

出力例

```
NAME                                DISPLAY                VERSION REPLACES PHASE
numaresources-operator.v4.13.2     numaresources-operator 4.13.2      Succeeded
```

6.2.2. Web コンソールを使用した NUMA Resources Operator のインストール

クラスター管理者は、Web コンソールを使用して NUMA Resources Operator をインストールできます。

手順

1. NUMA Resources Operator の namespace を作成します。
 - a. OpenShift Container Platform Web コンソールで、**Administration** → **Namespaces** をクリックします。
 - b. **Create Namespace** をクリックし、**Name** フィールドに **openshift-numresources** と入力して **Create** をクリックします。
2. NUMA Resources Operator をインストールします。
 - a. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
 - b. 利用可能な Operator のリストから **NUMA Resources Operator** を選択し、**Install** をクリックします。
 - c. **Installed Namespaces** フィールドで、**openshift-umaresources** namespace を選択して **Install** をクリックします。

3. オプション: NUMA Resources Operator が正常にインストールされたことを確認します。
 - a. **Operators** → **Installed Operators** ページに切り替えます。
 - b. **NUMA Resources Operator** が **openshift-umaresources** namespace にリストされ、**Status** が **InstallSucceeded** であることを確認します。



注記

インストール時に、Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operators** → **Installed Operators** ページに移動し、**Operator Subscriptions** および **Install Plans** タブで **Status** にエラーがあるかどうかを検査します。
- **Workloads** → **Pods** ページに移動し、**default** プロジェクトの Pod のログを確認します。

6.3. NUMA 対応ワークロードのスケジューリング

通常、遅延の影響を受けやすいワークロードを実行するクラスターは、ワークロードの遅延を最小限に抑え、パフォーマンスを最適化するのに役立つパフォーマンスプロファイルを備えています。NUMA 対応スケジューラーは、使用可能なノードの NUMA リソースと、ノードに適用されるパフォーマンスプロファイル設定に基づき、ワークロードをデプロイします。NUMA 対応デプロイメントとワークロードのパフォーマンスプロファイルを組み合わせることで、パフォーマンスを最大化するようにワークロードがスケジュールされます。

6.3.1. NUMAResourcesOperator カスタムリソースの作成

NUMA Resources Operator をインストールしたら、**NUMAResourcesOperator** カスタムリソース (CR) を作成します。この CR は、デーモンセットや API など、NUMA 対応スケジューラーをサポートするために必要なすべてのクラスターインフラストラクチャーをインストールするように NUMA Resources Operator に指示します。

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。
- NUMA Resources Operator をインストールしている。

手順

1. **NUMAResourcesOperator** カスタムリソースを作成します。
 - a. 以下の YAML を **nrop.yaml** ファイルに保存します。

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
```

```

name: numaresourcesoperator
spec:
  nodeGroups:
  - machineConfigPoolSelector:
      matchLabels:
        pools.operator.machineconfiguration.openshift.io/worker: ""

```

- b. 以下のコマンドを実行して、**NUMAResourcesOperator** CR を作成します。

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

検証

- 以下のコマンドを実行して、NUMA Resources Operator が正常にデプロイされたことを確認します。

```
$ oc get numaresourcesoperators.nodetopology.openshift.io
```

出力例

```

NAME                AGE
numaresourcesoperator 10m

```

6.3.2. NUMA 対応のセカンダリー Pod スケジューラーのデプロイ

NUMA Resources Operator をインストールしたら、次の手順を実行して NUMA 対応のセカンダリー Pod スケジューラーをデプロイします。

- パフォーマンスプロファイルを設定します。
- NUMA 対応のセカンダリースケジューラーをデプロイします。

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。
- 必要なマシン設定プールを作成している。
- NUMA Resources Operator をインストールしている。

手順

- PerformanceProfile** カスタムリソース (CR) を作成します。
 - 次の YAML を **nro-perfprof.yaml** ファイルに保存します。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: perfprof-nrop
spec:
  cpu: ①

```

```
isolated: "4-51,56-103"
reserved: "0,1,2,3,52,53,54,55"
nodeSelector:
  node-role.kubernetes.io/worker: ""
numa:
  topologyPolicy: single-numa-node
```

- 1 **cpu.isolated** および **cpu.reserved** 仕様は、分離および予約された CPU の範囲を定義します。CPU 設定の有効な値を入力します。パフォーマンスプロファイルの設定について、詳しくは [関連情報](#) セクションを参照してください。

- b. 次のコマンドを実行して、**PerformanceProfile** CR を作成します。

```
$ oc create -f nro-perfprof.yaml
```

出力例

```
performanceprofile.performance.openshift.io/perfprof-nrop created
```

2. NUMA 対応のカスタム Pod スケジューラーをデプロイする **NUMAResourcesScheduler** カスタムリソースを作成します。

- a. 以下の YAML を **nro-scheduler.yaml** ファイルに保存します。

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.13"
  cacheResyncPeriod: "5s" 1
```

- 1 スケジューラーキャッシュの同期間隔を秒単位の値で入力します。ほとんどの実装におけるこの値は、**5** が一般的です。



注記

- **cacheResyncPeriod** 仕様を有効にすると、NUMA Resource Operator は、ノード上の保留中のリソースを監視し、定義された間隔でスケジューラーキャッシュ内のこの情報を同期することで、より正確なりソース可用性を報告できます。これは、次善のスケジューリング決定が引き起こす **Topology Affinity Error** エラーを最小限に抑えるのにも役立ちます。間隔が短いほど、ネットワーク負荷が大きくなります。デフォルトでは、**cacheResyncPeriod** 仕様は無効になっています。
- **cacheResyncPeriod** 仕様の実装には、**NUMAResourcesOperator** CR の **PodsFingerprinting** 仕様の値を **Enabled** に設定する必要があります。

- b. 次のコマンドを実行して、**NUMAResourcesScheduler** CR を作成します。

```
$ oc create -f nro-scheduler.yaml
```

検証

1. 次のコマンドを実行して、パフォーマンスプロファイルが適用されたことを確認します。

```
$ oc describe performanceprofile <performance-profile-name>
```

2. 次のコマンドを実行して、必要なリソースが正常にデプロイされたことを確認します。

```
$ oc get all -n openshift-numaresources
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
pod/numaresources-controller-manager-7575848485-bns4s 1/1 Running 0 13m
pod/numaresourcesoperator-worker-dvj4n                2/2 Running 0 16m
pod/numaresourcesoperator-worker-lcg4t                2/2 Running 0 16m
pod/secondary-scheduler-56994cf6cf-7qf4q             1/1 Running 0 16m
NAME                                DESIRED CURRENT READY UP-TO-DATE
AVAILABLE NODE SELECTOR AGE
daemonset.apps/numaresourcesoperator-worker 2 2 2 2 2 node-
role.kubernetes.io/worker= 16m
NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/numaresources-controller-manager 1/1 1 1 13m
deployment.apps/secondary-scheduler 1/1 1 1 16m
NAME                                DESIRED CURRENT READY AGE
replicaset.apps/numaresources-controller-manager-7575848485 1 1 1 13m
replicaset.apps/secondary-scheduler-56994cf6cf 1 1 1 16m
```

関連情報

- [Performance Profile Creator の概要](#)。

6.3.3. NUMA 対応スケジューラーを使用したワークロードのスケジューリング

ワークロードを処理するために最低限必要なリソースを指定する **Deployment** CR を使用して、NUMA 対応スケジューラーでワークロードをスケジュールできます。

次のデプロイメント例では、サンプルワークロードに NUMA 対応のスケジューリングを使用します。

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。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

1. 次のコマンドを実行して、クラスターにデプロイされている NUMA 対応スケジューラーの名前を取得します。

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o json | jq '.status.schedulerName'
```

出力例

```
topo-aware-scheduler
```

2. **topo-aware-scheduler** という名前のスケジューラーを使用する **Deployment** CR を作成します。次に例を示します。
 - a. 以下の YAML を **nro-deployment.yaml** ファイルに保存します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: numa-deployment-1
  namespace: openshift-numaresources
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      schedulerName: topo-aware-scheduler 1
      containers:
        - name: ctrn
          image: quay.io/openshifttest/hello-openshift:openshift
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              memory: "100Mi"
              cpu: "10"
            requests:
              memory: "100Mi"
              cpu: "10"
        - name: ctrn2
          image: registry.access.redhat.com/rhel:latest
          imagePullPolicy: IfNotPresent
          command: ["/bin/sh", "-c"]
          args: [ "while true; do sleep 1h; done;" ]
          resources:
            limits:
              memory: "100Mi"
              cpu: "8"
            requests:
              memory: "100Mi"
              cpu: "8"
```

- 1 **schedulerName** は、クラスターにデプロイされている NUMA 対応のスケジューラーの名前 (**topo-aware-scheduler** など) と一致する必要があります。

- b. 次のコマンドを実行して、**Deployment** CR を作成します。

```
$ oc create -f nro-deployment.yaml
```

検証

1. デプロイメントが正常に行われたことを確認します。

```
$ oc get pods -n openshift-numaresources
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
numa-deployment-1-56954b7b46-pfgw8  2/2   Running 0      129m
numaresources-controller-manager-7575848485-bns4s  1/1   Running 0      15h
numaresourcesoperator-worker-dvj4n    2/2   Running 0      18h
numaresourcesoperator-worker-lcg4t    2/2   Running 0      16h
secondary-scheduler-56994cf6cf-7qf4q  1/1   Running 0      18h
```

2. 次のコマンドを実行して、**topo-aware-scheduler** がデプロイされた Pod をスケジューリングしていることを確認します。

```
$ oc describe pod numa-deployment-1-56954b7b46-pfgw8 -n openshift-numaresources
```

出力例

```
Events:
  Type Reason      Age From          Message
  ---- -
  Normal Scheduled    130m topo-aware-scheduler Successfully assigned openshift-numaresources/numa-deployment-1-56954b7b46-pfgw8 to compute-0.example.com
```



注記

スケジューリングに使用可能なリソースよりも多くのリソースを要求するデプロイメントは、**MinimumReplicasUnavailable** エラーで失敗します。必要なリソースが利用可能になると、デプロイメントは成功します。Pod は、必要なリソースが利用可能になるまで **Pending** 状態のままになります。

3. ノードに割り当てられる予定のリソースが一覧表示されていることを確認します。
- a. 次のコマンドを実行して、デプロイメント Pod を実行しているノードを特定します。このとき、<namespace> は **Deployment** CR で指定した namespace に置き換えます。

```
$ oc get pods -n <namespace> -o wide
```

出力例

```

NAME                                READY STATUS  RESTARTS  AGE  IP           NODE
NOMINATED NODE READINESS GATES
numa-deployment-1-65684f8fcc-bw4bw  0/2   Running  0      82m  10.128.2.50
worker-0 <none> <none>

```

- b. 次のコマンドを実行します。このとき、<node_name> はデプロイメント Pod を実行しているノードの名前に置き換えます。

```
$ oc describe noderesourcetopologies.topology.node.k8s.io <node_name>
```

出力例

```

...
Zones:
Costs:
  Name: node-0
  Value: 10
  Name: node-1
  Value: 21
Name: node-0
Resources:
  Allocatable: 39
  Available: 21 1
  Capacity: 40
  Name: cpu
  Allocatable: 6442450944
  Available: 6442450944
  Capacity: 6442450944
  Name: hugepages-1Gi
  Allocatable: 134217728
  Available: 134217728
  Capacity: 134217728
  Name: hugepages-2Mi
  Allocatable: 262415904768
  Available: 262206189568
  Capacity: 270146007040
  Name: memory
Type: Node

```

- 1** 保証された Pod に割り当てられたリソースが原因で、**Available** な容量が減少しています。

保証された Pod によって消費されるリソースは、**noderesourcetopologies.topology.node.k8s.io** にリスト表示されている使用可能なノードリソースから差し引かれます。

4. **Best-effort** または **Burstable** の サービス品質 (**qosClass**) を持つ Pod のリソース割り当てが、**noderesourcetopologies.topology.node.k8s.io** の NUMA ノードリソースに反映されていません。Pod の消費リソースがノードリソースの計算に反映されない場合は、Pod の **qosClass** が **Guaranteed** で、CPU 要求が 10 進値ではなく整数値であることを確認してください。次のコマンドを実行すると、Pod の **qosClass** が **Guaranteed** であることを確認できます。


```
$ oc get pod <pod_name> -n <pod_namespace> -o jsonpath="{ .status.qosClass }"
```

出力例

```
Guaranteed
```

6.4. 手動でのパフォーマンス設定による NUMA 対応ワークロードのスケジューリング

通常、遅延の影響を受けやすいワークロードを実行するクラスターは、ワークロードの遅延を最小限に抑え、パフォーマンスを最適化するのに役立つパフォーマンスプロファイルを備えています。ただし、パフォーマンスプロファイルを備えていない初期のクラスターで、NUMA 対応のワークロードをスケジューリングすることはできません。次のワークフローは、**KubeletConfig** リソースを使用してパフォーマンスを手動で設定できる初期のクラスターを特徴としています。これは、NUMA 対応ワークロードをスケジューリングするための一般的な環境ではありません。

6.4.1. 手動でのパフォーマンス設定による NUMAResourcesOperator カスタムリソースの作成

NUMA Resources Operator をインストールしたら、**NUMAResourcesOperator** カスタムリソース (CR) を作成します。この CR は、デーモンセットや API など、NUMA 対応スケジューラーをサポートするために必要なすべてのクラスターインフラストラクチャーをインストールするように NUMA Resources Operator に指示します。

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。
- NUMA Resources Operator をインストールしている。

手順

1. オプション: ワーカーノードのカスタム kubelet 設定を有効にする **MachineConfigPool** カスタムリソースを作成します。



注記

デフォルトでは、OpenShift Container Platform はクラスター内のワーカーノードの **MachineConfigPool** リソースを作成します。必要に応じて、カスタムの **MachineConfigPool** リソースを作成できます。

- a. 以下の YAML を **nro-machineconfig.yaml** ファイルに保存します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  labels:
    cnf-worker-tuning: enabled
    machineconfiguration.openshift.io/mco-built-in: ""
    pools.operator.machineconfiguration.openshift.io/worker: ""
```

```

name: worker
spec:
  machineConfigSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: worker
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""

```

- b. 以下のコマンドを実行して **MachineConfigPool** CR を作成します。

```
$ oc create -f nro-machineconfig.yaml
```

2. NUMAResourcesOperator カスタムリソースを作成します。

- a. 以下の YAML を **nrop.yaml** ファイルに保存します。

```

apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
  name: numaresourcesoperator
spec:
  nodeGroups:
  - machineConfigPoolSelector:
      matchLabels:
        pools.operator.machineconfiguration.openshift.io/worker: "" 1

```

- 1** 関連する **MachineConfigPool** CR でワーカーノードに適用されるラベルと一致する必要があります。

- b. 以下のコマンドを実行して、**NUMAResourcesOperator** CR を作成します。

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

検証

- 以下のコマンドを実行して、NUMA Resources Operator が正常にデプロイされたことを確認します。

```
$ oc get numaresourcesoperators.nodetopology.openshift.io
```

出力例

```

NAME                AGE
numaresourcesoperator 10m

```

6.4.2. 手動でのパフォーマンス設定による NUMA 対応セカンダリー Pod スケジューラーのデプロイ

NUMA Resources Operator をインストールしたら、次の手順を実行して NUMA 対応のセカンダリー Pod スケジューラーをデプロイします。

- 必要なマシンプロファイルの Pod アドミタンスポリシーを設定する
- 必要なマシン設定プールを作成する
- NUMA 対応のセカンダリースケジューラーをデプロイする

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ログインしている。
- NUMA Resources Operator をインストールしている。

手順

1. マシンプロファイルの Pod アドミタンスポリシーを設定する **KubeletConfig** カスタムリソースを作成します。
 - a. 以下の YAML を **nro-kubeletconfig.yaml** ファイルに保存します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cnf-worker-tuning
spec:
  machineConfigPoolSelector:
    matchLabels:
      cnf-worker-tuning: enabled
  kubeletConfig:
    cpuManagerPolicy: "static" ❶
    cpuManagerReconcilePeriod: "5s"
    reservedSystemCPUs: "0,1"
    memoryManagerPolicy: "Static" ❷
    evictionHard:
      memory.available: "100Mi"
    reservedMemory:
      - numaNode: 0
      limits:
        memory: "1124Mi"
    systemReserved:
      memory: "512Mi"
    topologyManagerPolicy: "single-numa-node" ❸
    topologyManagerScope: "pod"
```

- ❶ **cpuManagerPolicy** の場合、**static** は小文字の **s** を使用する必要があります。
- ❷ **memoryManagerPolicy** の場合、**Static** は大文字の **S** を使用する必要があります。
- ❸ **topologyManagerPolicy** は **single-numa-node** に設定する必要があります。

- b. 次のコマンドを実行して、**KubeletConfig** カスタムリソース (CR) を作成します。

```
$ oc create -f nro-kubeletconfig.yaml
```

2. NUMA 対応のカスタム Pod スケジューラーをデプロイする **NUMAResourcesScheduler** カスタムリソースを作成します。
 - a. 以下の YAML を **nro-scheduler.yaml** ファイルに保存します。

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.13"
  cacheResyncPeriod: "5s" ❶
```

- ❶ スケジューラーキャッシュの同期間隔を秒単位の値で入力します。ほとんどの実装におけるこの値は、**5** が一般的です。



注記

- **cacheResyncPeriod** 仕様を有効にすると、NUMA Resource Operator は、ノード上の保留中のリソースを監視し、定義された間隔でスケジューラーキャッシュ内のこの情報を同期することで、より正確なリソース可用性を報告できます。これは、次善のスケジューリング決定を引き起こす **Topology Affinity Error** エラーを最小限に抑えるのにも役立ちます。間隔が短いほど、ネットワーク負荷が大きくなります。デフォルトでは、**cacheResyncPeriod** 仕様は無効になっています。
- **cacheResyncPeriod** 仕様の実装には、**NUMAResourcesOperator** CR の **PodsFingerprinting** 仕様の値を **Enabled** に設定する必要があります。

- b. 次のコマンドを実行して、**NUMAResourcesScheduler** CR を作成します。

```
$ oc create -f nro-scheduler.yaml
```

検証

- 次のコマンドを実行して、必要なりソースが正常にデプロイされたことを確認します。

```
$ oc get all -n openshift-numaresources
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
pod/numaresources-controller-manager-7575848485-bns4s 1/1 Running 0 13m
pod/numaresourcesoperator-worker-dvj4n                2/2 Running 0 16m
pod/numaresourcesoperator-worker-lcg4t                2/2 Running 0 16m
pod/secondary-scheduler-56994cf6cf-7qf4q             1/1 Running 0 16m
NAME                                DESIRED CURRENT READY UP-TO-DATE
AVAILABLE NODE SELECTOR AGE
daemonset.apps/numaresourcesoperator-worker 2 2 2 2 2 node-
role.kubernetes.io/worker= 16m
NAME                                READY UP-TO-DATE AVAILABLE AGE
```

```

deployment.apps/numaresources-controller-manager 1/1 1 1 13m
deployment.apps/secondary-scheduler 1/1 1 1 16m
NAME DESIRED CURRENT READY AGE
replicaset.apps/numaresources-controller-manager-7575848485 1 1 1 13m
replicaset.apps/secondary-scheduler-56994cf6cf 1 1 1 16m

```

6.4.3. 手動でのパフォーマンス設定による NUMA 対応スケジューラーを使用したワークロードのスケジューリング

ワークロードを処理するために最低限必要なリソースを指定する **Deployment** CR を使用して、NUMA 対応スケジューラーでワークロードをスケジュールできます。

次のデプロイメント例では、サンプルワークロードに NUMA 対応のスケジューリングを使用します。

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。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

1. 次のコマンドを実行して、クラスターにデプロイされている NUMA 対応スケジューラーの名前を取得します。

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o json | jq '.status.schedulerName'
```

出力例

```
topo-aware-scheduler
```

2. **topo-aware-scheduler** という名前のスケジューラーを使用する **Deployment** CR を作成します。次に例を示します。
 - a. 以下の YAML を **nro-deployment.yaml** ファイルに保存します。

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: numa-deployment-1
  namespace: openshift-numaresources
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test

```

```
spec:
  schedulerName: topo-aware-scheduler 1
  containers:
  - name: ctrn
    image: quay.io/openshifttest/hello-openshift:openshift
    imagePullPolicy: IfNotPresent
  resources:
    limits:
      memory: "100Mi"
      cpu: "10"
    requests:
      memory: "100Mi"
      cpu: "10"
  - name: ctrn2
    image: registry.access.redhat.com/rhel:latest
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh", "-c"]
    args: [ "while true; do sleep 1h; done;" ]
  resources:
    limits:
      memory: "100Mi"
      cpu: "8"
    requests:
      memory: "100Mi"
      cpu: "8"
```

- 1** **schedulerName** は、クラスターにデプロイされている NUMA 対応のスケジューラーの名前 (**topo-aware-scheduler** など) と一致する必要があります。

- b. 次のコマンドを実行して、**Deployment** CR を作成します。

```
$ oc create -f nro-deployment.yaml
```

検証

1. デプロイメントが正常に行われたことを確認します。

```
$ oc get pods -n openshift-numaresources
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
numa-deployment-1-56954b7b46-pfgw8  2/2   Running 0      129m
numaresources-controller-manager-7575848485-bns4s  1/1   Running 0      15h
numaresourcesoperator-worker-dvj4n    2/2   Running 0      18h
numaresourcesoperator-worker-lcg4t    2/2   Running 0      16h
secondary-scheduler-56994cf6cf-7qf4q  1/1   Running 0      18h
```

2. 次のコマンドを実行して、**topo-aware-scheduler** がデプロイされた Pod をスケジューリングしていることを確認します。

```
$ oc describe pod numa-deployment-1-56954b7b46-pfgw8 -n openshift-numaresources
```

出力例

```
Events:
  Type Reason      Age From          Message
  ---- -
Normal Scheduled    130m topo-aware-scheduler Successfully assigned openshift-
numaresources/numa-deployment-1-56954b7b46-pfgw8 to compute-0.example.com
```



注記

スケジューリングに使用可能なリソースよりも多くのリソースを要求するデプロイメントは、**MinimumReplicasUnavailable** エラーで失敗します。必要なリソースが利用可能になると、デプロイメントは成功します。Pod は、必要なリソースが利用可能になるまで **Pending** 状態のままになります。

3. ノードに割り当てられる予定のリソースが一覧表示されていることを確認します。
 - a. 次のコマンドを実行して、デプロイメント Pod を実行しているノードを特定します。このとき、<namespace> は **Deployment** CR で指定した namespace に置き換えます。

```
$ oc get pods -n <namespace> -o wide
```

出力例

```
NAME                                READY STATUS RESTARTS AGE IP      NODE
NOMINATED NODE READINESS GATES
numa-deployment-1-65684f8fcc-bw4bw 0/2   Running 0      82m 10.128.2.50
worker-0 <none> <none>
```

- b. 次のコマンドを実行します。このとき、<node_name> はデプロイメント Pod を実行しているノードの名前に置き換えます。

```
$ oc describe noderesourcetopologies.topology.node.k8s.io <node_name>
```

出力例

```
...
Zones:
Costs:
  Name: node-0
  Value: 10
  Name: node-1
  Value: 21
Name: node-0
Resources:
  Allocatable: 39
  Available: 21 1
  Capacity: 40
  Name: cpu
  Allocatable: 6442450944
  Available: 6442450944
  Capacity: 6442450944
```

```

Name:      hugepages-1Gi
Allocatable: 134217728
Available: 134217728
Capacity:  134217728
Name:      hugepages-2Mi
Allocatable: 262415904768
Available: 262206189568
Capacity:  270146007040
Name:      memory
Type:      Node

```

- 1 保証された Pod に割り当てられたリソースが原因で、**Available** な容量が減少しています。

保証された Pod によって消費されるリソースは、**noderesourcetopologies.topology.node.k8s.io** にリスト表示されている使用可能なノードリソースから差し引かれます。

4. **Best-effort** または **Burstable** の サービス品質 (**qosClass**) を持つ Pod のリソース割り当てが、**noderesourcetopologies.topology.node.k8s.io** の NUMA ノードリソースに反映されていません。Pod の消費リソースがノードリソースの計算に反映されない場合は、Pod の **qosClass** が **Guaranteed** で、CPU 要求が 10 進値ではなく整数値であることを確認してください。次のコマンドを実行すると、Pod の **qosClass** が **Guaranteed** であることを確認できます。

```
$ oc get pod <pod_name> -n <pod_namespace> -o jsonpath="{.status.qosClass}"
```

出力例

```
Guaranteed
```

6.5. オプション: NUMA リソース更新のポーリング操作の設定

nodeGroup 内の NUMA Resources Operator によって制御されるデーモンは、リソースをポーリングして、利用可能な NUMA リソースに関する更新を取得します。**NUMAResourcesOperator** カスタムリソース (CR) で **spec.nodeGroups** 仕様を設定することで、これらのデーモンのポーリング操作を微調整できます。これにより、ポーリング操作の高度な制御が可能になります。これらの仕様を設定して、スケジューリング動作を改善し、最適ではないスケジューリング決定のトラブルシューティングを行います。

設定オプションは次のとおりです。

- **infoRefreshMode**: kubelet をポーリングするためのトリガー条件を決定します。NUMA Resources Operator は、結果として取得した情報を API サーバーに報告します。
- **infoRefreshPeriod**: ポーリング更新の間隔を決定します。
- **PodsFingerprinting**: ノード上で実行されている現在の Pod セットのポイントインタイム情報がポーリング更新で公開されるかどうかを決定します。



注記

PodsFingerprinting はデフォルトで有効になっています。**PodsFingerprinting** は、**NUMAResourcesScheduler** CR の **cacheResyncPeriod** 仕様の要件です。**cacheResyncPeriod** 仕様は、ノード上の保留中のリソースを監視することで、より正確なリソースの可用性を報告するのに役立ちます。

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。
- NUMA Resources Operator をインストールしている。

手順

- **NUMAResourcesOperator** CR で **spec.nodeGroups** 仕様を設定します。

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
  name: numaresourcesoperator
spec:
  nodeGroups:
  - config:
    infoRefreshMode: Periodic ①
    infoRefreshPeriod: 10s ②
    podsFingerprinting: Enabled ③
    name: worker
```

- ① 有効な値は **Periodic**、**Events**、**PeriodicAndEvents** です。**Periodic** を使用して、**infoRefreshPeriod** で定義した間隔で kubelet をポーリングします。**Events** を使用して、Pod のライフサイクルイベントごとに kubelet をポーリングします。両方のメソッドを有効にするには、**PeriodicAndEvents** を使用します。
- ② **Periodic** または **PeriodicAndEvents** リフレッシュモードのポーリング間隔を定義します。リフレッシュモードが **Events** の場合、このフィールドは無視されます。
- ③ 有効な値は **Enabled** と **Disabled** です。**NUMAResourcesScheduler** の **cacheResyncPeriod** 仕様では、**Enabled** への設定が必須です。

検証

1. NUMA Resources Operator をデプロイした後、次のコマンドを実行して、ノードグループ設定が適用されたことを検証します。

```
$ oc get numaresop numaresourcesoperator -o json | jq '.status'
```

出力例

```
...
```

```

"config": {
  "infoRefreshMode": "Periodic",
  "infoRefreshPeriod": "10s",
  "podsFingerprinting": "Enabled"
},
"name": "worker"
...

```

6.6. NUMA 対応スケジューリングのトラブルシューティング

NUMA 対応の Pod スケジューリングに関する一般的な問題をトラブルシューティングするには、次の手順を実行します。

前提条件

- OpenShift Container Platform CLI (**oc**) 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。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

1. 次のコマンドを実行して、**noderesourcetopologies** CRD がクラスターにデプロイされていることを確認します。

```
$ oc get crd | grep noderesourcetopologies
```

出力例

NAME	CREATED AT
noderesourcetopologies.topology.node.k8s.io	2022-01-18T08:28:06Z

2. 次のコマンドを実行して、NUMA 対応スケジューラー名が NUMA 対応ワークロードで指定された名前と一致することを確認します。

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o json | jq '.status.schedulerName'
```

出力例

```
topo-aware-scheduler
```

3. NUMA 対応のスケジュール可能なノードに **noderesourcetopologies** CR が適用され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 noderesourcetopologies.topology.node.k8s.io
```

出力例

NAME	AGE
compute-0.example.com	17h
compute-1.example.com	17h



注記

ノードの数は、マシン設定プール (**mcp**) ワーカー定義によって設定されているワーカーノードの数と等しくなければなりません。

- 次のコマンドを実行して、スケジューリング可能なすべてのノードの NUMA ゾーンの粒度を確認します。

```
$ oc get noderesourcetopologies.topology.node.k8s.io -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: topology.node.k8s.io/v1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoaware SchedWg/rte-update: periodic
    creationTimestamp: "2022-06-16T08:55:38Z"
    generation: 63760
    name: worker-0
    resourceVersion: "8450223"
    uid: 8b77be46-08c0-4074-927b-d49361471590
  topologyPolicies:
  - SingleNUMANodeContainerLevel
  zones:
  - costs:
    - name: node-0
      value: 10
    - name: node-1
      value: 21
    name: node-0
  resources:
  - allocatable: "38"
    available: "38"
    capacity: "40"
    name: cpu
  - allocatable: "134217728"
    available: "134217728"
    capacity: "134217728"
    name: hugepages-2Mi
  - allocatable: "262352048128"
    available: "262352048128"
    capacity: "270107316224"
    name: memory
  - allocatable: "6442450944"
    available: "6442450944"
    capacity: "6442450944"
    name: hugepages-1Gi
```

```

type: Node
- costs:
  - name: node-0
    value: 21
  - name: node-1
    value: 10
name: node-1
resources:
- allocatable: "268435456"
  available: "268435456"
  capacity: "268435456"
  name: hugepages-2Mi
- allocatable: "269231067136"
  available: "269231067136"
  capacity: "270573244416"
  name: memory
- allocatable: "40"
  available: "40"
  capacity: "40"
  name: cpu
- allocatable: "1073741824"
  available: "1073741824"
  capacity: "1073741824"
  name: hugepages-1Gi
type: Node
- apiVersion: topology.node.k8s.io/v1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoaware Schedwg/rte-update: periodic
    creationTimestamp: "2022-06-16T08:55:37Z"
    generation: 62061
    name: worker-1
    resourceVersion: "8450129"
    uid: e8659390-6f8d-4e67-9a51-1ea34bba1cc3
  topologyPolicies:
    - SingleNUMANodeContainerLevel
  zones: ①
- costs:
  - name: node-0
    value: 10
  - name: node-1
    value: 21
name: node-0
resources: ②
- allocatable: "38"
  available: "38"
  capacity: "40"
  name: cpu
- allocatable: "6442450944"
  available: "6442450944"
  capacity: "6442450944"
  name: hugepages-1Gi
- allocatable: "134217728"
  available: "134217728"
  capacity: "134217728"

```

```

name: hugepages-2Mi
- allocatable: "262391033856"
  available: "262391033856"
  capacity: "270146301952"
  name: memory
  type: Node
- costs:
  - name: node-0
    value: 21
  - name: node-1
    value: 10
  name: node-1
  resources:
  - allocatable: "40"
    available: "40"
    capacity: "40"
    name: cpu
  - allocatable: "1073741824"
    available: "1073741824"
    capacity: "1073741824"
    name: hugepages-1Gi
  - allocatable: "268435456"
    available: "268435456"
    capacity: "268435456"
    name: hugepages-2Mi
  - allocatable: "269192085504"
    available: "269192085504"
    capacity: "270534262784"
    name: memory
  type: Node
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""

```

- 1 **zones** 以下の各スタンプは、単一の NUMA ゾーンのリソースを記述しています。
- 2 **resources** は、NUMA ゾーンリソースの現在の状態を記述しています。 **items.zones.resources.available** 以下に記載されているリソースが、保証された各 Pod に割り当てられた排他的な NUMA ゾーンリソースに対応していることを確認します。

6.6.1. NUMA 対応スケジューラーログの確認

ログを確認して、NUMA 対応スケジューラーの問題をトラブルシューティングします。必要に応じて、**NUMAResourcesScheduler** リソースの **spec.logLevel** フィールドを変更して、スケジューラーのログレベルを上げることができます。許容値は **Normal**、**Debug**、および **Trace** で、**Trace** が最も詳細なオプションとなります。



注記

セカンダリースケジューラーのログレベルを変更するには、実行中のスケジューラーリソースを削除し、ログレベルを変更して再デプロイします。このダウンタイム中、スケジューラーは新しいワークロードのスケジューリングに使用できません。

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ログインしている。

手順

1. 現在実行中の **NUMAResourcesScheduler** リソースを削除します。
 - a. 次のコマンドを実行して、アクティブな **NUMAResourcesScheduler** を取得します。

```
$ oc get NUMAResourcesScheduler
```

出力例

```
NAME                AGE
numaresourcesscheduler 90m
```

- b. 次のコマンドを実行して、セカンダリースケジューラーリソースを削除します。

```
$ oc delete NUMAResourcesScheduler numaresourcesscheduler
```

出力例

```
numaresourcesscheduler.nodetopology.openshift.io "numaresourcesscheduler" deleted
```

2. 以下の YAML をファイル **nro-scheduler-debug.yaml** に保存します。この例では、ログレベルを **Debug** に変更します。

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.13"
  logLevel: Debug
```

3. 次のコマンドを実行して、更新された **Debug** ロギング **NUMAResourcesScheduler** リソースを作成します。

```
$ oc create -f nro-scheduler-debug.yaml
```

出力例

```
numaresourcesscheduler.nodetopology.openshift.io/numaresourcesscheduler created
```

検証手順

1. NUMA 対応スケジューラーが正常にデプロイされたことを確認します。
 - a. 次のコマンドを実行して、CRD が正常に作成されたことを確認します。

```
$ oc get crd | grep numaresourcesschedulers
```

出力例

```
NAME                                     CREATED AT
numaresourcesschedulers.nodetopology.openshift.io    2022-02-25T11:57:03Z
```

- b. 次のコマンドを実行して、新しいカスタムスケジューラーが使用可能であることを確認します。

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io
```

出力例

```
NAME          AGE
numaresourcesscheduler 3h26m
```

2. スケジューラーのログが増加したログレベルを示していることを確認します。

- a. 以下のコマンドを実行して、**openshift-numaresources** namespace で実行されている Pod のリストを取得します。

```
$ oc get pods -n openshift-numaresources
```

出力例

```
NAME                                     READY STATUS RESTARTS AGE
numaresources-controller-manager-d87d79587-76mrm 1/1 Running 0 46h
numaresourcesoperator-worker-5wm2k             2/2 Running 0 45h
numaresourcesoperator-worker-pb75c             2/2 Running 0 45h
secondary-scheduler-7976c4d466-qm4sc           1/1 Running 0 21m
```

- b. 次のコマンドを実行して、セカンダリースケジューラー Pod のログを取得します。

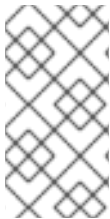
```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

出力例

```
...
I0223 11:04:55.614788    1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.Namespace total 11 items received
I0223 11:04:56.609114    1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.ReplicationController total 10 items received
I0223 11:05:22.626818    1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.StorageClass total 7 items received
I0223 11:05:31.610356    1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.PodDisruptionBudget total 7 items received
I0223 11:05:31.713032    1 eventhandlers.go:186] "Add event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
I0223 11:05:53.461016    1 eventhandlers.go:244] "Delete event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
```

6.6.2. リソーストポロジーエクスポーターのトラブルシューティング

対応する **resource-topology-exporter** ログを調べて、予期しない結果が発生している **noderesourcetopologies** オブジェクトをトラブルシューティングします。



注記

クラスター内の NUMA リソーストポロジーエクスポートインスタンスには、参照するノードの名前を付けることが推奨されます。たとえば、**worker** という名前のワーカーノードには、**worker** という対応する **noderesourcetopologies** オブジェクトがあるはず です。

前提条件

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

手順

1. NUMA Resources Operator によって管理されるデーモンセットを取得します。各 daemonset には、**NUMAResourcesOperator** CR 内に対応する **nodeGroup** があります。以下のコマンドを実行します。

```
$ oc get numaresourcesoperators.nodetopology.openshift.io numaresourcesoperator -o jsonpath="{.status.daemonsets[0]}"
```

出力例

```
{"name":"numaresourcesoperator-worker","namespace":"openshift-numaresources"}
```

2. 前のステップの **name** の値を使用して、対象となる daemonset のラベルを取得します。

```
$ oc get ds -n openshift-numaresources numaresourcesoperator-worker -o jsonpath="{.spec.selector.matchLabels}"
```

出力例

```
{"name":"resource-topology"}
```

3. 次のコマンドを実行して、**resource-topology** ラベルを使用して Pod を取得します。

```
$ oc get pods -n openshift-numaresources -l name=resource-topology -o wide
```

出力例

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
numaresourcesoperator-worker-5wm2k	2/2	Running	0	2d1h	10.135.0.64	compute-0.example.com
numaresourcesoperator-worker-pb75c	2/2	Running	0	2d1h	10.132.2.33	compute-1.example.com

4. トラブルシューティングしているノードに対応するワーカー Pod で実行されている **resource-topology-exporter** コンテナのログを調べます。以下のコマンドを実行します。

```
$ oc logs -n openshift-numaresources -c resource-topology-exporter numaresourcesoperator-worker-pb75c
```

出力例

```
10221 13:38:18.334140 1 main.go:206] using sysinfo:
reservedCpus: 0,1
reservedMemory:
"0": 1178599424
10221 13:38:18.334370 1 main.go:67] === System information ===
10221 13:38:18.334381 1 sysinfo.go:231] cpus: reserved "0-1"
10221 13:38:18.334493 1 sysinfo.go:237] cpus: online "0-103"
10221 13:38:18.546750 1 main.go:72]
cpus: allocatable "2-103"
hugepages-1Gi:
  numa cell 0 -> 6
  numa cell 1 -> 1
hugepages-2Mi:
  numa cell 0 -> 64
  numa cell 1 -> 128
memory:
  numa cell 0 -> 45758Mi
  numa cell 1 -> 48372Mi
```

6.6.3. 欠落しているリソーストポロジーエクスポート設定マップの修正

クラスター設定が正しく設定されていないクラスターに NUMA Resources Operator をインストールすると、場合によっては、Operator はアクティブとして表示されますが、リソーストポロジーエクスポート (RTE) デモンセット Pod のログには、RTE の設定が欠落していると表示されます。以下に例を示します。

```
Info: couldn't find configuration in "/etc/resource-topology-exporter/config.yaml"
```

このログメッセージは、必要な設定の **kubeletconfig** がクラスターに適切に適用されなかったため、RTE **configmap** が欠落していることを示しています。たとえば、次のクラスターには **numaresourcesoperator-worker configmap** カスタムリソース (CR) がありません。

```
$ oc get configmap
```

出力例

```
NAME                                DATA AGE
0e2a6bd3.openshift-kni.io          0    6d21h
kube-root-ca.crt                    1    6d21h
openshift-service-ca.crt            1    6d21h
topo-aware-scheduler-config         1    6d18h
```

正しく設定されたクラスターでは、**oc get configmap** は **numaresourcesoperator-worker configmap** CR も返します。

前提条件

- OpenShift Container Platform CLI (**oc**) 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。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

1. 次のコマンドを使用して、**kubeletconfig** の **spec.machineConfigPoolSelector.matchLabels** と **MachineConfigPool (mcp)** ワーカー CR の **metadata.labels** の値を比較します。

- a. 次のコマンドを実行して、**kubeletconfig** ラベルを確認します。

```
$ oc get kubeletconfig -o yaml
```

出力例

```
machineConfigPoolSelector:
  matchLabels:
    cnf-worker-tuning: enabled
```

- b. 次のコマンドを実行して、**mcp** ラベルを確認します。

```
$ oc get mcp worker -o yaml
```

出力例

```
labels:
  machineconfiguration.openshift.io/mco-built-in: ""
  pools.operator.machineconfiguration.openshift.io/worker: ""
```

cnf-worker-tuning: enabled ラベルが **MachineConfigPool** オブジェクトに存在しません。

2. **MachineConfigPool** CR を編集して、不足しているラベルを含めます。次に例を示します。

```
$ oc edit mcp worker -o yaml
```

出力例

```
labels:
  machineconfiguration.openshift.io/mco-built-in: ""
  pools.operator.machineconfiguration.openshift.io/worker: ""
  cnf-worker-tuning: enabled
```

3. ラベルの変更を適用し、クラスターが更新された設定を適用するのを待ちます。以下のコマンドを実行します。

検証

- 不足している **numaresourcesoperator-worker configmap** CR が適用されていることを確認します。

```
$ oc get configmap
```

出力例

```
NAME                               DATA AGE
0e2a6bd3.openshift-kni.io         0    6d21h
kube-root-ca.crt                   1    6d21h
numaresourcesoperator-worker       1     5m
openshift-service-ca.crt           1    6d21h
topo-aware-scheduler-config        1    6d18h
```

第7章 スケーラビリティとパフォーマンスの最適化

7.1. ストレージの最適化

ストレージを最適化すると、すべてのリソースでストレージの使用を最小限に抑えることができます。管理者は、ストレージを最適化することで、既存のストレージリソースが効率的に機能できるようにすることができます。

7.1.1. 利用可能な永続ストレージオプション

永続ストレージオプションについて理解し、OpenShift Container Platform 環境を最適化できるようにします。

表7.1利用可能なストレージオプション

ストレージタイプ	説明	例
ブロック	<ul style="list-style-type: none"> ブロックデバイスとしてオペレーティングシステムに公開されます。 ストレージを完全に制御し、ファイルシステムを通過してファイルの低いレベルで操作する必要のあるアプリケーションに適しています。 ストレージエリアネットワーク (SAN) とも呼ばれます。 共有できません。一度に1つのクライアントだけがこのタイプのエンドポイントをマウントできるという意味です。 	AWS EBS および VMware vSphere は、OpenShift Container Platform で永続ボリューム (PV) の動的なプロビジョニングをサポートします。
ファイル	<ul style="list-style-type: none"> マウントされるファイルシステムのエクスポートとして、OS に公開されます。 ネットワークアタッチストレージ (NAS) とも呼ばれます。 同時実行、レイテンシー、ファイルロックのメカニズムその他の各種機能は、プロトコルおよび実装、ベンダー、スケールによって大きく異なります。 	RHEL NFS、NetApp NFS ^[1] 、および Vendor NFS
オブジェクト	<ul style="list-style-type: none"> REST API エンドポイント経由でアクセスできます。 OpenShift イメージレジストリーで使用するように設定できます。 アプリケーションは、ドライバーをアプリケーションやコンテナに組み込む必要があります。 	AWS S3

1. NetApp NFS は Trident を使用する場合に動的 PV のプロビジョニングをサポートします。

7.1.2. 設定可能な推奨のストレージ技術

以下の表では、特定の OpenShift Container Platform クラスタアプリケーション向けに設定可能な推奨のストレージ技術についてまとめています。

表7.2 設定可能な推奨ストレージ技術

ストレージタイプ	ブロック	ファイル	オブジェクト
ROX ¹	はい ⁴	はい ⁴	はい
RWX ²	いいえ	はい	はい
レジストリー	設定可能	設定可能	推奨
スケーリングされたレジストリー	設定不可	設定可能	推奨
メトリクス ³	推奨	設定可能 ⁵	設定不可
Elasticsearch ログギング	推奨	設定可能 ⁶	サポート対象外 ⁶
Loki ログギング	設定不可	設定不可	推奨
アプリ	推奨	推奨	設定不可 ⁷

¹ **ReadOnlyMany**

² **ReadWriteMany**

³ Prometheus はメトリックに使用される基礎となるテクノロジーです。

⁴ これは、物理ディスク、VM 物理ディスク、VMDK、NFS 経由のループバック、AWS EBS、および Azure Disk には該当しません。

⁵ メトリックの場合、**ReadWriteMany (RWX)** アクセスモードのファイルストレージを信頼できる方法で使用することはできません。ファイルストレージを使用する場合、メトリクスと共に使用されるように設定される永続ボリューム要求 (PVC) で **RWX** アクセスモードを設定しないでください。

⁶ ログについては、ログストアの永続ストレージの設定セクションで推奨されるストレージソリューションを確認してください。NFS ストレージを永続ボリュームとして使用するか、Gluster などの NAS を介して使用すると、データが破損する可能性があります。したがって、NFS は、OpenShift Container Platform Logging の Elasticsearch ストレージおよび LokiStack ログストアではサポートされていません。ログストアごとに1つの永続的なボリュームタイプを使用する必要があります。

⁷ オブジェクトストレージは、OpenShift Container Platform の PV/PVC で消費されません。アプリは、オブジェクトストレージの REST API と統合する必要があります。



注記

スケーリングされたレジストリーは、2つ以上の Pod レプリカが実行されている OpenShift イメージレジストリーです。

7.1.2.1. 特定アプリケーションのストレージの推奨事項



重要

テストにより、NFS サーバーを Red Hat Enterprise Linux (RHEL) でコアサービスのストレージバックエンドとして使用することに関する問題が検出されています。これには、OpenShift Container レジストリーおよび Quay、メトリックストレージの Prometheus、およびロギングストレージの Elasticsearch が含まれます。そのため、コアサービスで使用される PV をサポートするために RHEL NFS を使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift Container Platform コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

7.1.2.1.1. レジストリー

スケーリングされていない/高可用性 (HA) OpenShift イメージレジストリークラスターのデプロイメントでは、次のようになります。

- ストレージ技術は、RWX アクセスモードをサポートする必要はありません。
- ストレージ技術は、リードアフターライト (Read-After-Write) の一貫性を確保する必要があります。
- 推奨されるストレージ技術はオブジェクトストレージであり、次はブロックストレージです。
- ファイルストレージは、実稼働ワークロードを使用した OpenShift イメージレジストリークラスターのデプロイメントには推奨されません。

7.1.2.1.2. スケーリングされたレジストリー

スケーリングされた/HA OpenShift イメージレジストリークラスターのデプロイメントでは、次のようになります。

- ストレージ技術は、RWX アクセスモードをサポートする必要があります。
- ストレージ技術は、リードアフターライト (Read-After-Write) の一貫性を確保する必要があります。
- 推奨されるストレージ技術はオブジェクトストレージです。
- Red Hat OpenShift Data Foundation (ODF)、Amazon Simple Storage Service (Amazon S3)、Google Cloud Storage (GCS)、Microsoft Azure Blob Storage、および OpenStack Swift がサポートされています。
- オブジェクトストレージは S3 または Swift に準拠する必要があります。
- vSphere やベアメタルインストールなどのクラウド以外のプラットフォームの場合、設定可能な技術はファイルストレージのみです。

- ブロックストレージは設定できません。

7.1.2.1.3. メトリクス

OpenShift Container Platform がホストするメトリックのクラスターデプロイメント:

- 推奨されるストレージ技術はブロックストレージです。
- オブジェクトストレージは設定できません。



重要

実稼働ワークロードがあるホスト型のメトリッククラスターデプロイメントにファイルストレージを使用することは推奨されません。

7.1.2.1.4. ロギング

OpenShift Container Platform がホストするロギングのクラスターデプロイメント:

- Loki Operator:
 - 推奨されるストレージテクノロジーは、S3 互換のオブジェクトストレージです。
 - ブロックストレージは設定できません。
- OpenShift Elasticsearch Operator:
 - 推奨されるストレージ技術はブロックストレージです。
 - オブジェクトストレージはサポートされていません。



注記

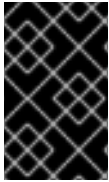
Logging バージョン 5.4.3 の時点で、OpenShift Elasticsearch Operator は非推奨であり、今後のリリースで削除される予定です。Red Hat は、この機能に対して現在のリリースライフサイクル中にバグ修正とサポートを提供しますが、拡張機能の提供はなく、この機能は今後削除される予定です。OpenShift Elasticsearch Operator を使用してデフォルトのログストレージを管理する代わりに、Loki Operator を使用できます。

7.1.2.1.5. アプリケーション

以下の例で説明されているように、アプリケーションのユースケースはアプリケーションごとに異なります。

- 動的な PV プロビジョニングをサポートするストレージ技術は、マウント時のレイテンシーが低く、ノードに関連付けられておらず、正常なクラスターをサポートします。
- アプリケーション開発者はアプリケーションのストレージ要件や、それがどのように提供されているストレージと共に機能するかを理解し、アプリケーションのスケールアップ時やストレージレイヤーと対話する際に問題が発生しないようにしておく必要があります。

7.1.2.2. 特定のアプリケーションおよびストレージの他の推奨事項



重要

etcd などの **Write** 集中型ワークロードで RAID 設定を使用することは推奨しません。RAID 設定で **etcd** を実行している場合、ワークロードでパフォーマンスの問題が発生するリスクがある可能性があります。

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder は ROX アクセスモードのユーザースペースで適切に機能する傾向があります。
- データベース: データベース (RDBMS、NoSQL DB など) は、専用のブロックストレージで最適に機能することが予想されます。
- etcd データベースには、大規模なクラスターを有効にするのに十分なストレージと十分なパフォーマンス容量が必要です。十分なストレージと高性能環境を確立するための監視およびベンチマークツールに関する情報は、**推奨される etcd プラクティス**に記載されています。

7.1.3. データストレージ管理

以下の表は、OpenShift Container Platform コンポーネントがデータを書き込むメインディレクトリーの概要を示しています。

表7.3 OpenShift Container Platform データを保存するメインディレクトリー

ディレクトリー	注記	サイジング	予想される拡張
/var/lib/etcd	データベースを保存する際に etcd ストレージに使用されます。	20 GB 未満。 データベースは、最大 8 GB まで拡張できます。	環境と共に徐々に拡張します。メタデータのみを格納します。 メモリーに 8 GB が追加されるたびに 20-25 GB を追加します。
/var/lib/containers	これは CRI-O ランタイムのマウントポイントです。アクティブなコンテナランタイム (Pod を含む) およびローカルイメージのストレージに使用されるストレージです。レジストリーストレージには使用されません。	16 GB メモリーの場合、1 ノードにつき 50 GB。 このサイジングは、クラスターの最小要件の決定には使用しないでください。 メモリーに 8 GB が追加されるたびに 20-25 GB を追加します。	拡張は実行中のコンテナの容量によって制限されます。
/var/log	すべてのコンポーネントのログファイルです。	10 から 30 GB。	ログファイルはすぐに拡張する可能性があります。サイズは拡張するディスク別に管理するか、ログローテーションを使用して管理できません。

ディレクトリー	注記	サイジング	予想される拡張
/var/lib/kubelet	Pod の一時ボリュームストレージです。これには、ランタイムにコンテナにマウントされる外部のすべての内容が含まれます。環境変数、kube シークレット、および永続ボリュームでサポートされていないデータボリュームが含まれます。	変動あり。	ストレージを必要とする Pod が永続ボリュームを使用している場合は最小になります。一時ストレージを使用する場合はすぐに拡張する可能性があります。
/var/log	すべてのコンポーネントのログファイルです。	10 から 30 GB。	ログファイルはすぐに拡張する可能性があります。サイズは拡張するディスク別に管理するか、ログローテーションを使用して管理できます。

7.1.4. Microsoft Azure のストレージパフォーマンスの最適化

OpenShift Container Platform と Kubernetes は、ディスクのパフォーマンスの影響を受けるため、特にコントロールプレーンノードの etcd には、より高速なストレージが推奨されます。

実稼働の Azure クラスターとワークロードが集中するクラスターの場合、コントロールプレーンマシンの仮想マシンオペレーティングシステムディスクは、テスト済みの推奨最小スループットである 5000 IOPS/200MBps を維持できなければなりません。このスループットは、P30 (最低 1 TiB Premium SSD) を使用することで実現できます。Azure および Azure Stack Hub の場合、ディスクパフォーマンスは SSD ディスクサイズに直接依存します。**Standard_D8s_v3** 仮想マシンまたは他の同様のマシンタイプでサポートされるスループットと 5000 IOPS の目標を達成するには、少なくとも P30 ディスクが必要です。

データ読み取り時のレイテンシーを低く抑え、高い IOPS およびスループットを実現するには、ホストのキャッシュを **ReadOnly** に設定する必要があります。仮想マシンメモリーまたはローカル SSD ディスクに存在するキャッシュからのデータの読み取りは、blob ストレージにあるディスクからの読み取りよりもはるかに高速です。

7.1.5. 関連情報

- [Elasticsearch ログストアの設定](#)

7.2. ルーティングの最適化

OpenShift Container Platform HAProxy ルーターは、パフォーマンスを最適化するためにスケーリングまたは設定できます。

7.2.1. ベースライン Ingress コントローラー (ルーター) のパフォーマンス

OpenShift Container Platform Ingress コントローラー (ルーター) は、ルートとインGRESSを使用して設定されたアプリケーションとサービスのインGRESSトラフィックのインGRESSポイントです。

1秒に処理される HTTP 要求について、単一の HAProxy ルーターを評価する場合に、パフォーマンスは多くの要因により左右されます。特に以下が含まれます。

- HTTP keep-alive/close モード
- ルートタイプ
- TLS セッション再開のクライアントサポート
- ターゲットルートごとの同時接続数
- ターゲットルート数
- バックエンドサーバーのページサイズ
- 基礎となるインフラストラクチャー (ネットワーク/SDN ソリューション、CPU など)

特定の環境でのパフォーマンスは異なりますが、Red Hat ラボはサイズが 4 vCPU/16GB RAM のパブリッククラウドインスタンスでテストしています。1kB 静的ページを提供するバックエンドで終端する 100 ルートを処理する単一の HAProxy ルーターは、1秒あたりに以下の数のトランザクションを処理できます。

HTTP keep-alive モードのシナリオの場合:

暗号化	LoadBalancerService	HostNetwork
なし	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

HTTP close (keep-alive なし) のシナリオの場合:

暗号化	LoadBalancerService	HostNetwork
なし	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

デフォルトの Ingress Controller 設定は、**spec.tuningOptions.threadCount** フィールドを **4** に設定して、使用されました。Load Balancer Service と Host Network という 2つの異なるエンドポイント公開

戦略がテストされました。TLS セッション再開は暗号化ルートについて使用されています。HTTP keep-alive では、1台の HAProxy ルーターで、8kB という小さなページサイズで 1Gbit の NIC を飽和させることができます。

最新のプロセッサが搭載されたベアメタルで実行する場合は、上記のパブリッククラウドインスタンスのパフォーマンスの約 2 倍のパフォーマンスになることを予想できます。このオーバーヘッドは、パブリッククラウドにある仮想化レイヤーにより発生し、プライベートクラウドベースの仮想化にも多くの場合、該当します。以下の表は、ルーターの背後で使用するアプリケーション数についてのガイドです。

アプリケーション数	アプリケーションタイプ
5-10	静的なファイル/Web サーバーまたはキャッシュプロキシー
100-1000	動的なコンテンツを生成するアプリケーション

通常、HAProxy は、使用しているテクノロジーに応じて、最大 1000 個のアプリケーションのルートをサポートできます。Ingress コントローラーのパフォーマンスは、言語や静的コンテンツと動的コンテンツの違いを含め、その背後にあるアプリケーションの機能およびパフォーマンスによって制限される可能性があります。

Ingress またはルーターのシャード化は、アプリケーションに対してより多くのルートを提供するために使用され、ルーティング層の水平スケーリングに役立ちます。

Ingress のシャード化についての詳細は、[ルートラベルを使用した Ingress コントローラーのシャード化の設定](#) および [namespace ラベルを使用した Ingress コントローラーのシャード化の設定](#) を参照してください。

スレッドの [Ingress Controller スレッド数の設定](#)、タイムアウトの [Ingress Controller 設定パラメーター](#)、および Ingress Controller 仕様のその他のチューニング設定で提供されている情報を使用して、Ingress Controller デプロイメントを変更できます。

7.2.2. Ingress コントローラー (ルーター) liveness、readiness、および startup プローブの設定

クラスター管理者は、OpenShift Container Platform Ingress Controller (ルーター) によって管理されるルーター展開の kubelet の活性、準備、およびスタートアッププローブのタイムアウト値を設定できます。ルーターの liveness および readiness プローブは、デフォルトのタイムアウト値である 1 秒を使用します。これは、ネットワークまたはランタイムのパフォーマンスが著しく低下している場合には短すぎます。プローブのタイムアウトにより、アプリケーション接続を中断する不要なルーターの再起動が発生する可能性があります。より大きなタイムアウト値を設定する機能により、不要で不要な再起動のリスクを減らすことができます。

ルーターコンテナの **livenessProbe**、**readinessProbe**、および **startupProbe** パラメーターの **timeoutSeconds** 値を更新できます。

パラメーター	説明
livenessProbe	livenessProbe は、Pod が停止していて再起動が必要かどうかを kubelet に報告します。

パラメーター	説明
readinessProbe	readinessProbe は、Pod が正常かどうかを報告します。準備プローブが異常な Pod を報告すると、kubelet は Pod をトラフィックを受け入れる準備ができていないものとしてマークします。その後、その Pod のエンドポイントは準備ができていないとマークされ、このステータスが kube-proxy に伝播されます。ロードバランサーが設定されたクラウドプラットフォームでは、kube-proxy はクラウドロードバランサーと通信して、その Pod を持つノードにトラフィックを送信しません。
startupProbe	startupProbe は、kubelet がルーターの活性と準備のプローブの送信を開始する前に、ルーター Pod の初期化に最大 2 分を与えます。この初期化時間により、多くのルートまたはエンドポイントを持つルーターが時期尚早に再起動するのを防ぐことができます。



重要

タイムアウト設定オプションは、問題を回避するために使用できる高度なチューニング手法です。ただし、これらの問題は最終的に診断する必要があり、プローブがタイムアウトする原因となる問題については、サポートケースまたは [Jira issue](#) を開く必要があります。

次の例は、デフォルトのルーター展開に直接パッチを適用して、活性プローブと準備プローブに 5 秒のタイムアウトを設定する方法を示しています。

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5},"readinessProbe":{"timeoutSeconds":5}]}}}}}'
```

検証

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
Liveness: http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
Readiness: http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1 #failure=3
```

7.2.3. HAProxy リロード間隔の設定

ルートまたはルートに関連付けられたエンドポイントを更新すると、OpenShift Container Platform ルーターは HAProxy の設定を更新します。次に、HAProxy は更新された設定をリロードして、これらの変更を有効にします。HAProxy がリロードすると、更新された設定を使用して新しい接続を処理する新しいプロセスが生成されます。

HAProxy は、それらの接続がすべて閉じられるまで、既存の接続を処理するために古いプロセスを実行し続けます。古いプロセスの接続が長く続くと、これらのプロセスはリソースを蓄積して消費する可能性があります。

デフォルトの最小 HAProxy リロード間隔は 5 秒です。**spec.tuningOptions.reloadInterval** フィールドを使用して Ingress コントローラーを設定し、より長い最小リロード間隔を設定できます。



警告

最小 HAProxy リロード間隔に大きな値を設定すると、ルートとそのエンドポイントの更新を監視する際にレイテンシーが発生する可能性があります。リスクを軽減するには、更新の許容レイテンシーよりも大きな値を設定しないようにしてください。

手順

- 次のコマンドを実行して、Ingress コントローラーのデフォルト最小 HAProxy リロード間隔を 15 秒に変更します。

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
```

7.3. ネットワークの最適化

[OpenShift SDN](#) は OpenvSwitch、VXLAN (Virtual extensible LAN) トンネル、OpenFlow ルール、iptables を使用します。このネットワークは、ジャンボフレーム、ネットワークインターフェイスコントローラー (NIC) オフロード、マルチキュー、および ethtool 設定を使用して調整できます。

[OVN-Kubernetes](#) は、トンネルプロトコルとして VXLAN ではなく Geneve (Generic Network Virtualization Encapsulation) を使用します。

VXLAN は、4096 から 1600 万以上にネットワーク数が増え、物理ネットワーク全体で階層 2 の接続が追加されるなど、VLAN での利点が提供されます。これにより、異なるシステム上で実行されている場合でも、サービスの背後にある Pod すべてが相互に通信できるようになります。

VXLAN は、User Datagram Protocol (UDP) パケットにトンネル化されたトラフィックをすべてカプセル化しますが、CPU 使用率が上昇してしまいます。これらの外部および内部パケットは、移動中にデータが破損しないようにするために通常のチェックサムルールの対象になります。これらの外部および内部パケットはどちらも、移動中にデータが破損しないように通常のチェックサムルールの対象になります。CPU のパフォーマンスによっては、この追加の処理オーバーヘッドによってスループットが減り、従来の非オーバーレイネットワークと比較してレイテンシーが高くなります。

クラウド、仮想マシン、ベアメタルの CPU パフォーマンスでは、1 Gbps をはるかに超えるネットワークスループットを処理できます。10 または 40 Gbps などの高い帯域幅のリンクを使用する場合には、パフォーマンスが低減する場合があります。これは、VXLAN ベースの環境では既知の問題で、コンテナや OpenShift Container Platform 固有の問題ではありません。VXLAN トンネルに依存するネットワークも、VXLAN 実装により同様のパフォーマンスになります。

1 Gbps 以上にするには、以下を実行してください。

- Border Gateway Protocol (BGP) など、異なるルーティング技術を実装するネットワークプラグインを評価する。
- VXLAN オフロード対応のネットワークアダプターを使用します。VXLAN オフロードは、システムの CPU から、パケットのチェックサム計算と関連の CPU オーバーヘッドを、ネットワークアダプター上の専用のハードウェアに移動します。これにより、CPU サイクルを Pod やアプリケーションで使用できるように開放し、ネットワークインフラストラクチャーの帯域幅すべてをユーザーは活用できるようになります。

VXLAN オフロードはレイテンシーを短縮しません。ただし、CPU の使用率はレイテンシーテストでも削減されます。

7.3.1. ネットワークでの MTU の最適化

重要な Maximum Transmission Unit (MTU) が 2 つあります。1 つはネットワークインターフェイスコントローラー (NIC) MTU で、もう 1 つはクラスターネットワーク MTU です。

NIC MTU は OpenShift Container Platform のインストール時にのみ設定されます。MTU は、お使いのネットワークの NIC でサポートされる最大の値以下でなければなりません。スループットを最適化する場合は、可能な限り大きい値を選択します。レイテンシーを最低限に抑えるために最適化するには、より小さい値を選択します。

OpenShift SDN ネットワークプラグインオーバーレイ MTU は、NIC MTU よりも少なくとも 50 バイト小さくする必要があります。これは、SDN オーバーレイのヘッダーに相当します。したがって、通常のイーサネットネットワークでは、これを **1450** に設定する必要があります。ジャンボフレームイーサネットネットワークでは、これを **8950** に設定する必要があります。これらの値は、NIC に設定された MTU に基づいて、Cluster Network Operator によって自動的に設定される必要があります。したがって、クラスター管理者は通常、これらの値を更新しません。Amazon Web Services (AWS) およびペアメタル環境は、ジャンボフレームイーサネットネットワークをサポートします。この設定は、特に伝送制御プロトコル (TCP) のスループットに役立ちます。

OVN および Geneve については、MTU は最低でも NIC MTU より 100 バイト少なくなければなりません。



注記

この 50 バイトのオーバーレイヘッダーは、OpenShift SDN ネットワークプラグインに関連します。他の SDN ソリューションの場合はこの値を若干変動させる必要があります。

7.3.2. 大規模なクラスターのインストールに推奨されるプラクティス

大規模なクラスターをインストールする場合や、クラスターを大規模なノード数に拡張する場合、クラスターをインストールする前に、**install-config.yaml** ファイルに適宜クラスターネットワーク **cidr** を設定します。

```
networking:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  machineNetwork:
    - cidr: 10.0.0.0/16
  networkType: OVNKubernetes
  serviceNetwork:
    - 172.30.0.0/16
```

クラスターのサイズが 500 を超える場合、デフォルトのクラスターネットワーク **cidr 10.128.0.0/14** を使用することはできません。500 ノードを超えるノード数にするには、**10.128.0.0/12** または **10.128.0.0/10** に設定する必要があります。

7.3.3. IPsec の影響

ノードホストの暗号化、復号化に CPU 機能が使用されるので、使用する IP セキュリティシステムにかかわらず、ノードのスループットおよび CPU 使用率の両方でのパフォーマンスに影響があります。

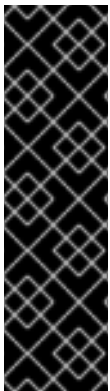
IPSec は、NIC に到達する前に IP ペイロードレベルでトラフィックを暗号化して、NIC オフロードに使用されてしまう可能性のあるフィールドを保護します。つまり、IPSec が有効な場合には、NIC アクセラレーション機能を使用できない場合があります、スループットの減少、CPU 使用率の上昇につながります。

7.3.4. 関連情報

- [高度なネットワーク設定パラメーターの変更](#)
- [OVN-Kubernetes ネットワークプラグインの設定パラメーター](#)
- [OpenShift SDN ネットワークプラグインの設定パラメーター](#)
- [ワーカレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスターの安定性の向上](#)

7.4. マウント NAMESPACE のカプセル化による CPU 使用率の最適化

マウント namespace のカプセル化を使用して kubelet および CRI-O プロセスにプライベート namespace を提供することで、OpenShift Container Platform クラスターでの CPU 使用率を最適化できます。これにより、機能に違いはなく、systemd が使用するクラスター CPU リソースが削減されません。



重要

マウント namespace のカプセル化は、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。

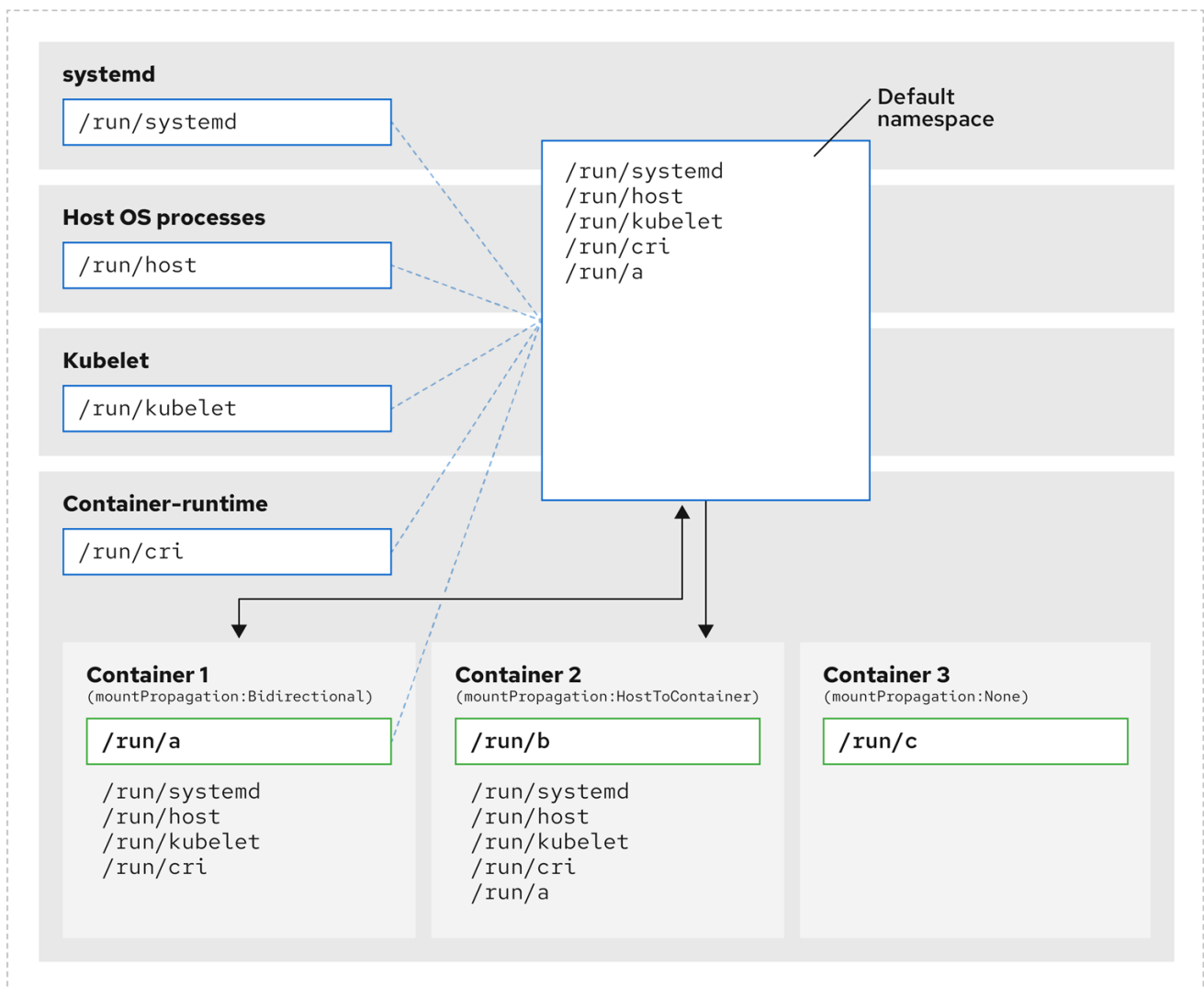
7.4.1. マウント namespace のカプセル化

マウント namespace は、異なる namespace のプロセスが互いのファイルを表示できないように、マウントポイントを分離するために使用されます。カプセル化は、Kubernetes マウント namespace を、ホストオペレーティングシステムによって常にスキャンされない別の場所に移動するプロセスです。

ホストオペレーティングシステムは systemd を使用して、すべてのマウント namespace (標準の Linux マウントと、Kubernetes が操作に使用する多数のマウントの両方) を常にスキャンします。kubelet と CRI-O の現在の実装はどちらも、すべてのコンテナランタイムと kubelet マウントポイントに最上位の namespace を使用します。ただし、これらのコンテナ固有のマウントポイントをプライベート namespace にカプセル化すると、systemd のオーバーヘッドが削減され、機能に違いはありません。CRI-O と kubelet の両方に個別のマウント namespace を使用すると、systemd または他のホスト OS の相互作用からコンテナ固有のマウントをカプセル化できます。

CPU の大幅な最適化を潜在的に達成するこの機能は、すべての OpenShift Container Platform 管理者が利用できるようになりました。カプセル化は、Kubernetes 固有のマウントポイントの特権のないユーザーによる検査から安全な場所に保存することで、セキュリティを向上させることもできます。

次の図は、カプセル化の前後の Kubernetes インストールを示しています。どちらのシナリオも、双方向、ホストからコンテナ、およびなしのマウント伝搬設定を持つコンテナの例を示しています。

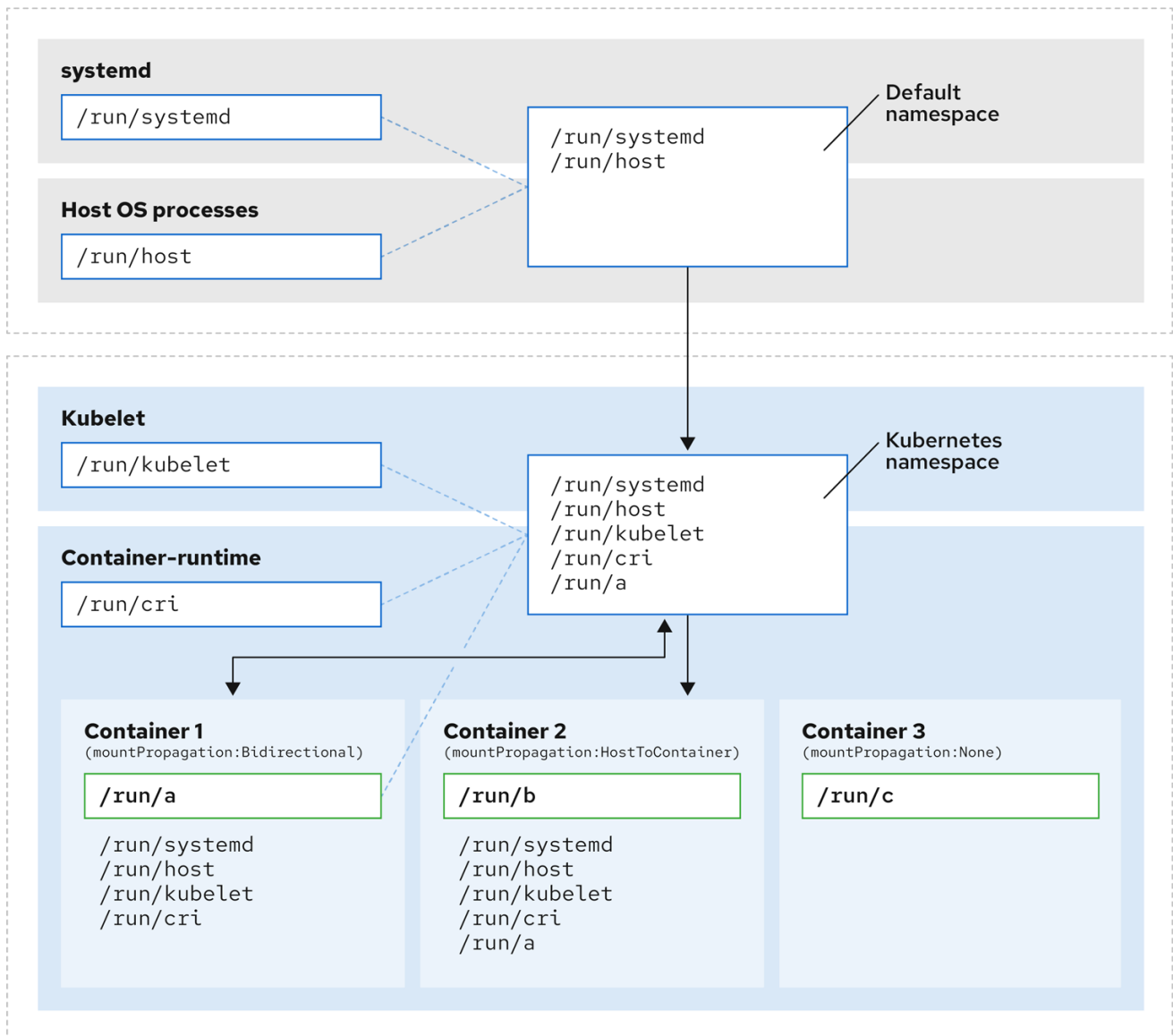


290_OpenShift_1122

ここでは、systemd、ホストオペレーティングシステムプロセス、kubelet、およびコンテナランタイムが単一のマウント namespace を共有していることがわかります。

- systemd、ホストオペレーティングシステムプロセス、kubelet、およびコンテナランタイムはそれぞれ、すべてのマウントポイントにアクセスして可視化できます。
- コンテナ1は、双方向のマウント伝達で設定され、systemd およびホストマウント、kubelet および CRI-O マウントにアクセスできます。`/run/a` などのコンテナ1で開始されたマウントは、systemd、ホスト OS プロセス、kubelet、コンテナランタイム、およびホストからコンテナへのまたは双方向のマウント伝達が設定されている他のコンテナ (コンテナ2のように) に表示されます。
- ホストからコンテナへのマウント伝達で設定されたコンテナ2は、systemd およびホストマウント、kubelet および CRI-O マウントにアクセスできます。`/run/b` などのコンテナ2で発生したマウントは、他のコンテキストからは見えません。
- マウント伝達なしで設定されたコンテナ3には、外部マウントポイントが表示されません。`/run/c` などのコンテナ3で開始されたマウントは、他のコンテキストからは見えません。

次の図は、カプセル化後のシステム状態を示しています。

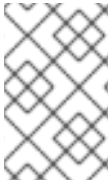


290_OpenShift_1122

- メインの systemd プロセスは、Kubernetes 固有のマウントポイントの不要なスキャンに専念しなくなりました。systemd 固有のホストマウントポイントのみを監視します。
- ホストオペレーティングシステムプロセスは、systemd およびホストマウントポイントにのみアクセスできます。
- CRI-O と kubelet の両方に個別のマウント namespace を使用すると、すべてのコンテナ固有のマウントが systemd または他のホスト OS の対話から完全に分離されます。
- コンテナ 1 の動作は変更されていませんが、/run/a などのコンテナが作成するマウントが systemd またはホスト OS プロセスから認識されなくなります。kubelet、CRI-O、およびホストからコンテナまたは双方向のマウント伝達が設定されている他のコンテナ（コンテナ 2 など）からは引き続き表示されます。
- コンテナ 2 とコンテナ 3 の動作は変更されていません。

7.4.2. マウント namespace のカプセル化の設定

クラスターがより少ないリソースオーバーヘッドで実行されるように、マウント namespace のカプセル化を設定できます。



注記

マウント namespace のカプセル化はテクノロジープレビュー機能であり、デフォルトでは無効になっています。これを使用するには、機能を手動で有効に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ログインしている。

手順

1. 次の YAML を使用して、**mount_namespace_config.yaml** という名前のファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-kubens-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kubens.service
---
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-kubens-worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kubens.service
```

2. 次のコマンドを実行して、マウント namespace **MachineConfig** CR を適用します。

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

出力例

```
machineconfig.machineconfiguration.openshift.io/99-kubens-master created
machineconfig.machineconfiguration.openshift.io/99-kubens-worker created
```

3. **MachineConfig** CR がクラスターに適用されるまで、最大 30 分かかる場合があります。次のコマンドを実行して、**MachineConfig** CR のステータスをチェックできます。

```
$ oc get mcp
```

出力例

```
NAME CONFIG UPDATED UPDATING DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT AGE
master rendered-master-03d4bc4befb0f4ed3566a2c8f7636751 False True False
3 0 0 0 45m
worker rendered-worker-10577f6ab0117ed1825f8af2ac687ddf False True False
3 1 1
```

4. 次のコマンドを実行した後、**MachineConfig** CR がすべてのコントロールプレーンとワーカーノードに正常に適用されるまで待ちます。

```
$ oc wait --for=condition=Updated mcp --all --timeout=30m
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/master condition met
machineconfigpool.machineconfiguration.openshift.io/worker condition met
```

検証

クラスターホストのカプセル化を確認するには、次のコマンドを実行します。

1. クラスターホストへのデバッグシェルを開きます。

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

2. **chroot** セッションを開きます。

```
sh-4.4# chroot /host
```

3. systemd マウント namespace を確認します。

```
sh-4.4# readlink /proc/1/ns/mnt
```

出力例

```
mnt:[4026531953]
```

4. kubelet マウント namespace をチェックします。

```
sh-4.4# readlink /proc/$(pgrep kubelet)/ns/mnt
```

出力例

```
mnt:[4026531840]
```

5. CRI-O マウント namespace を確認します。

```
sh-4.4# readlink /proc/$(pgrep cri-o)/ns/mnt
```

出力例

```
mnt:[4026531840]
```

これらのコマンドは、systemd、kubelet、およびコンテナランタイムに関連付けられたマウント namespace を返します。OpenShift Container Platform では、コンテナランタイムは CRI-O です。

上記の例のように、systemd が kubelet および CRI-O とは異なるマウント namespace にある場合、カプセル化が有効になります。3つのプロセスすべてが同じマウント namespace にある場合、カプセル化は有効ではありません。

7.4.3. カプセル化された namespace の検査

Red Hat Enterprise Linux CoreOS (RHCOS) で利用可能な **kubensenter** スクリプトを使用して、デバッグまたは監査の目的でクラスターホストオペレーティングシステムの Kubernetes 固有のマウントポイントを検査できます。

クラスターホストへの SSH シェルセッションは、デフォルトの namespace にあります。SSH シェルプロンプトで Kubernetes 固有のマウントポイントを検査するには、ルートとして **kubensenter** スクリプトを実行する必要があります。**kubensenter** スクリプトは、マウントカプセル化の状態を認識しており、カプセル化が有効になっていない場合でも安全に実行できます。



注記

oc debug リモートシェルセッションは、デフォルトで Kubernetes namespace 内で開始されます。**oc debug** を使用する場合、マウントポイントを検査するために **kubensenter** を実行する必要はありません。

カプセル化機能が有効になっていない場合、**kubensenter findmnt** コマンドと **findmnt** コマンドは、**oc debug** セッションで実行されているか SSH シェルプロンプトで実行されているかに関係なく、同じ出力を返します。

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。
- クラスターホストへの SSH アクセスを設定しました。

手順

1. クラスターホストへのリモート SSH シェルを開きます。以下に例を示します。

```
$ ssh core@<node_name>
```

2. root ユーザーとして、提供された **kubensenter** スクリプトを使用してコマンドを実行します。Kubernetes namespace 内で単一のコマンドを実行するには、コマンドと任意の引数を **kubensenter** スクリプトに提供します。たとえば、Kubernetes namespace 内で **findmnt** コマンドを実行するには、次のコマンドを実行します。

```
[core@control-plane-1 ~]$ sudo kubensenter findmnt
```

出力例

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
TARGET          SOURCE          FSTYPE  OPTIONS
/
/dev/sda4[ostree/deploy/rhcos/deploy/32074f0e8e5ec453e56f5a8a7bc9347eaa4172349ceab9c22b709d9d71a3f4b0.0]
|
|              xfs
rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota
|              shm          tmpfs
...
```

3. Kubernetes namespace 内で新しいインタラクティブシェルを開始するには、引数を指定せずに **kubensenter** スクリプトを実行します。

```
[core@control-plane-1 ~]$ sudo kubensenter
```

出力例

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
```

7.4.4. カプセル化された namespace で追加サービスを実行する

ホスト OS で実行する機能に依存し、kubelet、CRI-O、またはコンテナ自体によって作成されたマウントポイントを表示できる監視ツールは、これらのマウントポイントを表示するためにコンテナマウント namespace に入る必要があります。OpenShift Container Platform に付属する **kubensenter** スクリプトは、Kubernetes マウントポイント内で別のコマンドを実行し、既存のツールを適応させるために使用できます。

kubensenter スクリプトは、マウントカプセル化機能の状態を認識しており、カプセル化が有効になっていない場合でも安全に実行できます。その場合、スクリプトはデフォルトのマウント namespace で提供されたコマンドを実行します。

たとえば、systemd サービスを新しい Kubernetes マウント namespace 内で実行する必要がある場合は、サービスファイルを編集し、**kubensenter** で **ExecStart=** コマンドラインを使用します。

```
[Unit]
Description=Example service
[Service]
ExecStart=/usr/bin/kubensenter /path/to/original/command arg1 arg2
```

7.4.5. 関連情報

- [namespace](#) とは
- [nsenter](#) を使用して namespace 内のコンテナを管理する
- [MachineConfig](#)

第8章 ベアメタルホストの管理

OpenShift Container Platform をベアメタルクラスターにインストールする場合、クラスターに存在するベアメタルホストの **machine** および **machineset** カスタムリソース (CR) を使用して、ベアメタルノードをプロビジョニングし、管理できます。

8.1. ベアメタルホストおよびノードについて

Red Hat Enterprise Linux CoreOS (RHCOS) ベアメタルホストをクラスター内のノードとしてプロビジョニングするには、まずベアメタルホストハードウェアに対応する **MachineSet** カスタムリソース (CR) オブジェクトを作成します。ベアメタルホストコンピュータマシンセットは、お使いの設定に固有のインフラストラクチャーコンポーネントを記述します。特定の Kubernetes ラベルをこれらのコンピュータマシンセットに適用してから、インフラストラクチャーコンポーネントを更新して、それらのマシンでのみ実行されるようにします。

Machine CR は、**metal3.io/autoscale-to-hosts** アノテーションを含む関連する **MachineSet** をスケールアップする際に自動的に作成されます。OpenShift Container Platform は **Machine** CR を使用して、**MachineSet** CR で指定されるホストに対応するベアメタルノードをプロビジョニングします。

8.2. ベアメタルホストのメンテナンス

OpenShift Container Platform Web コンソールからクラスター内のベアメタルホストの詳細を維持することができます。 **Compute** → **Bare Metal Hosts** に移動し、 **Actions** ドロップダウンメニューからタスクを選択します。ここでは、BMC の詳細、ホストの起動 MAC アドレス、電源管理の有効化などの項目を管理できます。また、ホストのネットワークインターフェイスおよびドライブの詳細を確認することもできます。

ベアメタルホストをメンテナンスモードに移行できます。ホストをメンテナンスモードに移行すると、スケジューラーはすべての管理ワークロードに対応するベアメタルノードから移動します。新しいワークロードは、メンテナンスモードの間はスケジュールされません。

Web コンソールでベアメタルホストのプロビジョニングを解除することができます。ホストのプロビジョニング解除により以下のアクションが実行されます。

1. ベアメタルホスト CR に **cluster.k8s.io/delete-machine: true** のアノテーションを付けます。
2. 関連するコンピュータマシンセットをスケールダウンします



注記

デーモンセットおよび管理対象外の静的 Pod を別のノードに最初に移動することなく、ホストの電源をオフにすると、サービスの中断やデータの損失が生じる場合があります。

関連情報

- [コンピュータマシンのベアメタルへの追加](#)

8.2.1. Web コンソールを使用したベアメタルホストのクラスターへの追加

Web コンソールのクラスターにベアメタルホストを追加できます。

前提条件

- RHCOS クラスターのベアメタルへのインストール
- **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. Web コンソールで、**Compute** → **Bare Metal Hosts**に移動します。
2. **Add Host** → **New with Dialog** を選択します。
3. 新規ベアメタルホストの一意的な名前を指定します。
4. **Boot MAC address** を設定します。
5. **Baseboard Management Console (BMC) Address** を設定します。
6. ホストのベースボード管理コントローラー (BMC) のユーザー認証情報を入力します。
7. 作成後にホストの電源をオンにすることを選択し、**Create** を選択します。
8. 利用可能なベアメタルホストの数に一致するようにレプリカ数をスケールアップします。**Compute** → **MachineSets** に移動し、**Actions** ドロップダウンメニューから **Edit Machine count** を選択してクラスター内のマシンレプリカ数を増やします。



注記

oc scale コマンドおよび適切なベアメタルコンピュートマシンセットを使用して、ベアメタルノードの数を管理することもできます。

8.2.2. Web コンソールの YAML を使用したベアメタルホストのクラスターへの追加

ベアメタルホストを記述する YAML ファイルを使用して、Web コンソールのクラスターにベアメタルホストを追加できます。

前提条件

- クラスターで使用するために RHCOS コンピュートマシンをベアメタルインフラストラクチャーに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。
- ベアメタルホストの **Secret** CR を作成します。

手順

1. Web コンソールで、**Compute** → **Bare Metal Hosts**に移動します。
2. **Add Host** → **New from YAML** を選択します。
3. 以下の YAML をコピーして貼り付け、ホストの詳細で関連フィールドを変更します。

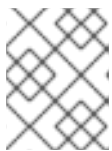
```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <bare_metal_host_name>
```



```
spec:
  online: true
  bmc:
    address: <bmc_address>
    credentialsName: <secret_credentials_name> ❶
    disableCertificateVerification: True ❷
    bootMACAddress: <host_boot_mac_address>
```

- ❶ **credentialsName** は有効な **Secret** CR を参照する必要があります。 **baremetal-operator** は、 **credentialsName** で参照される有効な **Secret** なしに、ベアメタルホストを管理できません。シークレットの詳細および作成方法については、[シークレットの理解](#) を参照してください。
- ❷ **disableCertificateVerification** を **true** に設定すると、クラスターとベースボード管理コントローラー (BMC) の間の TLS ホスト検証が無効になります。

4. **Create** を選択して YAML を保存し、新規ベアメタルホストを作成します。
5. 利用可能なベアメタルホストの数に一致するようにレプリカ数をスケールアップします。 **Compute** → **MachineSets** に移動し、 **Actions** ドロップダウンメニューから **Edit Machine count** を選択してクラスター内のマシン数を増やします。



注記

oc scale コマンドおよび適切なベアメタルコンピュートマシンセットを使用して、ベアメタルノードの数を管理することもできます。

8.2.3. 利用可能なベアメタルホストの数へのマシンの自動スケーリング

利用可能な **BareMetalHost** オブジェクトの数に一致する **Machine** オブジェクトの数を自動的に作成するには、 **metal3.io/autoscale-to-hosts** アノテーションを **MachineSet** オブジェクトに追加します。

前提条件

- クラスターで使用する RHCOS ベアメタルコンピュートマシンをインストールし、対応する **BareMetalHost** オブジェクトを作成します。
- OpenShift Container Platform CLI (**oc**) 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。

手順

1. **metal3.io/autoscale-to-hosts** アノテーションを追加して、自動スケーリング用に設定するコンピュートマシンセットにアノテーションを付けます。 **<machineset>** をコンピュートマシンセットの名前に置き換えます。

```
$ oc annotate machineset <machineset> -n openshift-machine-api 'metal3.io/autoscale-to-hosts=<any_value>'
```

新しいスケーリングされたマシンが起動するまで待ちます。



注記

BareMetalHost オブジェクトを使用してクラスター内にマシンを作成し、その後ラベルまたはセレクターが **BareMetalHost** で変更される場合、**BareMetalHost** オブジェクトは **Machine** オブジェクトが作成された **MachineSet** に対して引き続きカウントされません。

8.2.4. プロビジョナーノードからのベアメタルホストの削除

特定の状況では、プロビジョナーノードからベアメタルホストを一時的に削除する場合があります。たとえば、OpenShift Container Platform 管理コンソールを使用して、または Machine Config Pool の更新の結果として、ベアメタルホストの再起動がトリガーされたプロビジョニング中に、OpenShift Container Platform は統合された Dell Remote Access Controller (iDrac) にログインし、ジョブキューの削除を発行します。

利用可能な **BareMetalHost** オブジェクトの数と一致する数の **Machine** オブジェクトを管理しないようにするには、**baremetalhost.metal3.io/detached** アノテーションを **MachineSet** オブジェクトに追加します。



注記

このアノテーションは、**Provisioned**、**ExternallyProvisioned**、または **Ready/Available** 状態の **BareMetalHost** オブジェクトに対してのみ効果があります。

前提条件

- クラスターで使用する RHCOS ベアメタルコンピュートマシンをインストールし、対応する **BareMetalHost** オブジェクトを作成します。
- OpenShift Container Platform CLI (**oc**) 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。

手順

1. プロビジョナーノードから削除するコンピューティングマシンセットに、**baremetalhost.metal3.io/detached** アノテーションを追加してアノテーションを付けます。

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached'
```

新しいマシンが起動するまで待ちます。



注記

BareMetalHost オブジェクトを使用してクラスター内にマシンを作成し、その後ラベルまたはセレクターが **BareMetalHost** で変更される場合、**BareMetalHost** オブジェクトは **Machine** オブジェクトが作成された **MachineSet** に対して引き続きカウントされます。

2. プロビジョニングのユースケースでは、次のコマンドを使用して、再起動が完了した後にアノテーションを削除します。

```
$ oc annotate machineset <machineset> -n openshift-machine-api  
'baremetalhost.metal3.io/detached-'
```

関連情報

- [クラスタの拡張](#)
- [ベアメタル上の MachineHealthCheck](#)

第9章 BARE METAL EVENT RELAY を使用したベアメタルイベントのモニタリング



重要

Bare Metal Event Relay はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品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。

9.1. ベアメタルイベント

Bare Metal Event Relay を使用して、OpenShift Container Platform クラスターで実行されるアプリケーションを、基礎となるベアメタルホストで生成されるイベントにサブスクライブします。Redfish サービスは、ノードでイベントをパブリッシュし、サブスクライブされたアプリケーションに高度なメッセージキューでイベントを送信します。

ベアメタルイベントは、Distributed Management Task Force (DMTF) のガイダンスに基づいて開発されたオープン Redfish 標準に基づいています。Redfish は、REST API を使用してセキュアな業界標準プロトコルを提供します。このプロトコルは、分散された、コンバージドまたはソフトウェア定義のソースおよびインフラストラクチャーの管理に使用されます。

Redfish から公開されるハードウェア関連のイベントには、以下が含まれます。

- 温度制限の違反
- サーバステータス
- fan ステータス

Bare Metal Event Relay Operator をデプロイし、アプリケーションをサービスにサブスクライブして、ベアメタルイベントの使用を開始します。Bare Metal Event Relay Operator は Redfish ベアメタルイベントサービスのライフサイクルをインストールし、管理します。



注記

Bare Metal Event Relay は、ベアメタルインフラストラクチャーにプロビジョニングされる単一ノードクラスターの Redfish 対応デバイスでのみ機能します。

9.2. ベアメタルイベントの仕組み

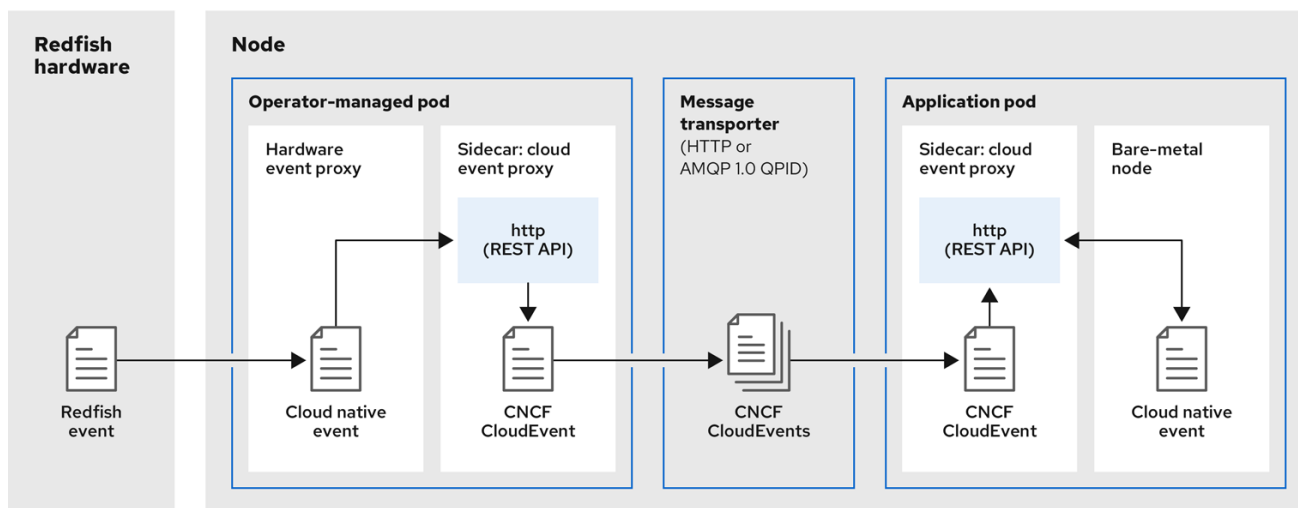
Bare Metal Event Relay により、ベアメタルクラスターで実行されるアプリケーションが Redfish ハードウェアの変更や障害に迅速に対応することができます。たとえば、温度のしきい値の違反、fan の障害、ディスク損失、電源停止、メモリー障害などが挙げられます。これらのハードウェアイベントは、HTTP トランスポートまたは AMQP メカニズムを使用して配信されます。メッセージングサービスのレイテンシーは 10 ミリ秒から 20 ミリ秒です。

Bare Metal Event Relay により、ハードウェアイベントでパブリッシュ - サブスクライブサービスを使用できます。アプリケーションは、REST API を使用してイベントをサブスクライブできます。Bare Metal Event Relay は、Redfish OpenAPI v1.8 以降に準拠するハードウェアをサポートします。

9.2.1. Bare Metal Event Relay データフロー

以下の図は、ベアメタルイベントのデータフローの例を示しています。

図9.1 Bare Metal Event Relay データフロー



319_OpenShift_0323

9.2.1.1. Operator 管理の Pod

Operator はカスタムリソースを使用して、**HardwareEvent** CR を使用して Bare Metal Event Relay およびそのコンポーネントが含まれる Pod を管理します。

9.2.1.2. Bare Metal イベントリレー

起動時に、Bare Metal Event Relay は Redfish API をクエリーし、カスタムレジストリーを含むすべてのメッセージレジストリーをダウンロードします。その後、Bare Metal Event Relay は Redfish ハードウェアからサブスクライブされたイベントを受信し始めます。

Bare Metal Event Relay により、ベアメタルクラスターで実行されるアプリケーションが Redfish ハードウェアの変更や障害に迅速に対応することができます。たとえば、温度のしきい値の違反、fan の障害、ディスク損失、電源停止、メモリー障害などが挙げられます。イベントは **HardwareEvent** CR を使用してレポートされます。

9.2.1.3. クラウドネイティブイベント

クラウドネイティブイベント (CNE) は、イベントデータの形式を定義する REST API 仕様です。

9.2.1.4. CNCF CloudEvents

CloudEvents は、イベントデータの形式を定義するために Cloud Native Computing Foundation (CNCF) によって開発されたベンダーに依存しない仕様です。

9.2.1.5. HTTP トランスポートまたは AMQP ディスパッチルーター

HTTP トランスポートまたは AMQP ディスパッチャーは、パブリッシャーとサブスクライバー間のメッセージ配信サービスを行います。



注記

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

9.2.1.6. クラウドイベントプロキシサイドカー

クラウドイベントプロキシサイドカーコンテナイメージは O-RAN API 仕様をベースとしており、ハードウェアイベントのパブリッシュ - サブスクライブイベントフレームワークを提供します。

9.2.2. サービスを解析する Redfish メッセージ

Bare Metal Event Relay は Redfish イベントを処理する他に、**Message** プロパティなしでイベントのメッセージ解析を提供します。プロキシは、起動時にハードウェアからベンダー固有のレジストリーを含むすべての Redfish メッセージブローカーをダウンロードします。イベントに **Message** プロパティが含まれていない場合、プロキシは Redfish メッセージレジストリーを使用して **Message** プロパティおよび **Resolution** プロパティを作成し、イベントをクラウドイベントフレームワークに渡す前にイベントに追加します。このサービスにより、Redfish イベントでメッセージサイズが小さくなり、送信レイテンシーが短縮されます。

9.2.3. CLI を使用した Bare Metal Event リレーのインストール

クラスター管理者は、CLI を使用して Bare Metal Event Relay Operator をインストールできます。

前提条件

- RedFish 対応ベースボード管理コントローラー (BMC) を持つノードでベアメタルハードウェアにインストールされるクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. Bare Metal Event Relay の namespace を作成します。
 - a. 以下の YAML を **bare-metal-events-namespace.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-bare-metal-events
  labels:
    name: openshift-bare-metal-events
    openshift.io/cluster-monitoring: "true"
```

- b. **namespace** CR を作成します。

```
$ oc create -f bare-metal-events-namespace.yaml
```

2. Bare Metal Event Relay Operator の Operator グループを作成します。

- a. 以下の YAML を **bare-metal-events-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: bare-metal-event-relay-group
  namespace: openshift-bare-metal-events
spec:
  targetNamespaces:
    - openshift-bare-metal-events
```

- b. **OperatorGroup** CR を作成します。

```
$ oc create -f bare-metal-events-operatorgroup.yaml
```

3. Bare Metal Event Relay にサブスクライブします。

- a. 以下の YAML を **bare-metal-events-sub.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bare-metal-event-relay-subscription
  namespace: openshift-bare-metal-events
spec:
  channel: "stable"
  name: bare-metal-event-relay
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- b. **Subscription** CR を作成します。

```
$ oc create -f bare-metal-events-sub.yaml
```

検証

Bare Metal Event Relay Operator がインストールされていることを確認するには、以下のコマンドを実行します。

```
$ oc get csv -n openshift-bare-metal-events -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

9.2.4. Web コンソールを使用した Bare Metal Event リレーのインストール

クラスター管理者は、Web コンソールを使用して Bare Metal Event Relay Operator をインストールできます。

前提条件

- RedFish 対応ベースボード管理コントローラー (BMC) を持つノードでベアメタルハードウェアにインストールされるクラスター。
- **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 コンソールを使用して Bare Metal Event Relay をインストールします。
 - a. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
 - b. 利用可能な Operator のリストから **Bare Metal Event Relay**を選択し、**Install** をクリックします。
 - c. **Install Operator**ページで、**Namespace** を選択または作成し、**openshift-bare-metal-events** を選択して、**Install** をクリックします。

検証

オプション: 以下のチェックを実行して、Operator が正常にインストールされていることを確認できます。

1. **Operators** → **Installed Operators** ページに切り替えます。
2. **Status** が **InstallSucceeded** の状態で、**Bare Metal Event Relay**がプロジェクトにリスト表示されていることを確認します。



注記

インストール時に、Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operators** → **Installed Operators** ページに移動し、**Operator Subscriptions** および **Install Plans** タブで **Status** にエラーがあるかどうかを検査します。
- **Workloads** → **Pods** ページに移動し、プロジェクト namespace で Pod のログを確認します。

9.3. AMQ メッセージングバスのインストール

ノードのパブリッシャーとサブスクライバー間で Redfish ベアメタルイベント通知を渡すには、ノード上でローカルを実行するように AMQ メッセージングバスをインストールし、設定できます。これは、クラスターで使用するために AMQ Interconnect Operator をインストールして行います。



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 Container Platform CLI (**oc**) 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。

手順

- AMQ Interconnect Operator を独自の **amq-interconnect** namespace にインストールします。[AMQ Interconnect Operator のインストール](#) について参照してください。

検証

1. AMQ Interconnect Operator が利用可能で、必要な Pod が実行されていることを確認します。

```
$ oc get pods -n amq-interconnect
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
amq-interconnect-645db76c76-k8ghs	1/1	Running	0	23h
interconnect-operator-5cb5fc7cc-4v7qm	1/1	Running	0	23h

2. 必要な **bare-metal-event-relay** ベアメタルイベントプロデューサー Pod が **openshift-bare-metal-events** namespace で実行されていることを確認します。

```
$ oc get pods -n openshift-bare-metal-events
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
hw-event-proxy-operator-controller-manager-74d5649b7c-dzgtl	2/2	Running	0	25s

9.4. クラスターノードの REDFISH BMC ベアメタルイベントのサブスクライブ

ノードの **BMCEventSubscription** カスタムリソース (CR) の作成、イベント用の **HardwareEvent** CR の作成、BMC の **Secret** CR の作成を行うことで、クラスター内のノードで生成される Redfish BMC イベントにサブスクライブできます。

9.4.1. ベアメタルイベントのサブスクライブ

ベースボード管理コントローラー (BMC) を設定して、ベアメタルイベントを OpenShift Container Platform クラスターで実行されているサブスクライブされたアプリケーションに送信できます。Redfish ベアメタルイベントの例には、デバイス温度の増加やデバイスの削除が含まれます。REST API を使用して、アプリケーションをベアメタルイベントにサブスクライブします。



重要

BMCEventSubscription カスタムリソース (CR) は、Redfish をサポートし、ベンダーインターフェイスが **redfish** または **idrac-redfish** に設定されている物理ハードウェアのみ作成できます。



注記

BMCEventSubscription CR を使用して事前定義された Redfish イベントにサブスクライブします。Redfish 標準は、特定のアラートおよびしきい値を作成するオプションを提供しません。例えば、エンクロージャーの温度が摂氏 40 度を超えたときにアラートイベントを受け取るには、ベンダーの推奨に従ってイベントを手動で設定する必要があります。

BMCEventSubscription CR を使用してノードのベアメタルイベントをサブスクライブするには、以下の手順を行います。

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ログインしている。
- BMC のユーザー名およびパスワードを取得します。
- クラスターに Redfish が有効な Baseboard Management Controller (BMC) を持つベアメタルノードをデプロイし、BMC で Redfish イベントを有効にします。



注記

特定のハードウェアで Redfish イベントを有効にすることは、この情報の対象範囲外です。特定のハードウェアの Redfish イベントを有効にする方法は、BMC の製造元のドキュメントを参照してください。

手順

1. 以下の **curl** コマンドを実行して、ノードのハードウェアで Redfish **EventService** が有効になっていることを確認します。

```
$ curl https://<bmc_ip_address>/redfish/v1/EventService --insecure -H 'Content-Type: application/json' -u "<bmc_username>:<password>"
```

ここでは、以下のようになります。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

出力例

```

{
  "@odata.context": "/redfish/v1/$metadata#EventService.EventService",
  "@odata.id": "/redfish/v1/EventService",
  "@odata.type": "#EventService.v1_0_2.EventService",
  "Actions": {
    "#EventService.SubmitTestEvent": {
      "EventType@Redfish.AllowableValues": ["StatusChange", "ResourceUpdated",
"ResourceAdded", "ResourceRemoved", "Alert"],
      "target": "/redfish/v1/EventService/Actions/EventService.SubmitTestEvent"
    }
  },
  "DeliveryRetryAttempts": 3,
  "DeliveryRetryIntervalSeconds": 30,
  "Description": "Event Service represents the properties for the service",
  "EventTypesForSubscription": ["StatusChange", "ResourceUpdated", "ResourceAdded",
"ResourceRemoved", "Alert"],
  "EventTypesForSubscription@odata.count": 5,
  "Id": "EventService",
  "Name": "Event Service",
  "ServiceEnabled": true,
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "Subscriptions": {
    "@odata.id": "/redfish/v1/EventService/Subscriptions"
  }
}

```

2. 以下のコマンドを実行して、クラスターの Bare Metal Event Relay サービスのルートを取得します。

```
$ oc get route -n openshift-bare-metal-events
```

出力例

NAME	HOST/PORT	PATH	SERVICES
PORT	TERMINATION	WILDCARD	
hw-event-proxy	hw-event-proxy	openshift-bare-metal-events.apps.compute-1.example.com	hw-event-proxy-service 9087 edge None

3. **BMCEventSubscription** リソースを作成し、Redfish イベントにサブスクライブします。
 - a. 以下の YAML を **bmc_sub.yaml** ファイルに保存します。

```

apiVersion: metal3.io/v1alpha1
kind: BMCEventSubscription
metadata:
  name: sub-01
  namespace: openshift-machine-api
spec:

```

```

hostName: <hostname> ①
destination: <proxy_service_url> ②
context: "

```

- ① Redfish イベントが生成されるワーカーノードの名前または UUID を指定します。
- ② ベアメタルイベントプロキシーサービスを指定します (例: <https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook>)。

b. **BMCEventSubscription** CR を作成します。

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

4. オプション: BMC イベントサブスクリプションを削除するには、以下のコマンドを実行します。

```
$ oc delete -f bmc_sub.yaml
```

5. オプション:**BMCEventSubscription** CR を作成せずに Redfish イベントサブスクリプションを手動で作成するには、BMC のユーザー名およびパスワードを指定して以下の **curl** コマンドを実行します。

```

$ curl -i -k -X POST -H "Content-Type: application/json" -d '{"Destination":
"https://<proxy_service_url>", "Protocol": "Redfish", "EventTypes": ["Alert"], "Context":
"root"}' -u <bmc_username>:<password>
'https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions' -v

```

ここでは、以下ようになります。

proxy_service_url

ベアメタルイベントプロキシーサービスです (例: <https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook>)。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

出力例

```

HTTP/1.1 201 Created
Server: AMI MegaRAC Redfish Service
Location: /redfish/v1/EventService/Subscriptions/1
Allow: GET, POST
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: X-Auth-Token
Access-Control-Allow-Headers: X-Auth-Token
Access-Control-Allow-Credentials: true
Cache-Control: no-cache, must-revalidate
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>; rel=describedby
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>
Link: </redfish/v1/EventService/Subscriptions>; path=
ETag: "1651135676"
Content-Type: application/json; charset=UTF-8

```

```

OData-Version: 4.0
Content-Length: 614
Date: Thu, 28 Apr 2022 08:47:57 GMT

```

9.4.2. curl を使用した Redfish ベアメタルイベントサブスクリプションのクエリー

一部のハードウェアベンダーは Redfish ハードウェアイベントサブスクリプションの量を制限します。**curl** を使用して Redfish イベントサブスクリプションの数をクエリーできます。

前提条件

- BMC のユーザー名およびパスワードを取得します。
- クラスタに Redfish が有効な Baseboard Management Controller (BMC) を持つベアメタルノードをデプロイし、BMC で Redfish ハードウェアイベントを有効にします。

手順

1. 以下の **curl** コマンドを実行して、BMC の現在のサブスクリプションを確認します。

```

$ curl --globoff -H "Content-Type: application/json" -k -X GET --user <bmc_username>:
<password> https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions

```

ここでは、以下のようになります。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

出力例

```

% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 435 100 435 0 0 399 0 0:00:01 0:00:01 --:--:-- 399
{
  "@odata.context":
"/redfish/v1/$metadata#EventDestinationCollection.EventDestinationCollection",
  "@odata.etag": "",
  1651137375 "",
  "@odata.id": "/redfish/v1/EventService/Subscriptions",
  "@odata.type": "#EventDestinationCollection.EventDestinationCollection",
  "Description": "Collection for Event Subscriptions",
  "Members": [
    {
      "@odata.id": "/redfish/v1/EventService/Subscriptions/1"
    }
  ],
  "Members@odata.count": 1,
  "Name": "Event Subscriptions Collection"
}

```

この例では、サブスクリプションが1つ設定されています (/redfish/v1/EventService/Subscriptions/1)。

2. オプション: **curl** で /redfish/v1/EventService/Subscriptions/1 サブスクリプションを削除するには、BMC のユーザー名およびパスワードを指定して以下のコマンドを実行します。

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\n" -k -u <bmc_username>:<password >-
H "Content-Type: application/json" -d '{} ' -X DELETE
https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions/1
```

ここでは、以下のようになります。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

9.4.3. ベアメタルイベントおよびシークレット CR の作成

ベアメタルイベントの使用を開始するには、Redfish ハードウェアが存在するホストの **HardwareEvent** カスタムリソース (CR) を作成します。ハードウェアイベントと障害は **hw-event-proxy** ログに報告されます。

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。
- Bare Metal Event Relay をインストールしている。
- BMC Redfish ハードウェア用の **BMCEventSubscription** CR を作成している。

手順

1. **HardwareEvent** カスタムリソース (CR) を作成します。



注記

複数の **HardwareEvent** リソースは許可されません。

- a. 以下の YAML を **hw-event.yaml** ファイルに保存します。

```
apiVersion: "event.redhat-cne.org/v1alpha1"
kind: "HardwareEvent"
metadata:
  name: "hardware-event"
spec:
  nodeSelector:
    node-role.kubernetes.io/hw-event: "" ❶
  logLevel: "debug" ❷
  msgParserTimeout: "10" ❸
```

- ❶ 必須。 **nodeSelector** フィールドを使用して、指定されたラベルを持つノードをターゲットにします (例: **node-role.kubernetes.io/hw-event: ""**)。



注記

OpenShift Container Platform 4.13 以降でベアメタルイベントに HTTP トランスポートを使用する場合、**HardwareEvent** リソースの **spec.transportHost** フィールドを設定する必要はありません。ベアメタルイベントに AMQP トランスポートを使用する場合にのみ **transportHost** を設定します。

- 2 オプション: デフォルト値は **debug** です。 **hw-event-proxy** ログでログレベルを設定します。 **fatal**、 **error**、 **warning**、 **info**、 **debug**、 **trace** のログレベルを利用できません。
- 3 オプション: Message Parser のタイムアウト値をミリ秒単位で設定します。メッセージ解析要求がタイムアウト期間内に応答しない場合には、元のハードウェアイベントメッセージはクラウドネイティブイベントフレームワークに渡されます。デフォルト値は 10 です。

- b. クラスタで **HardwareEvent** CR を適用します。

```
$ oc create -f hardware-event.yaml
```

2. BMC ユーザー名およびパスワード **Secret** CR を作成します。これにより、ハードウェアイベントプロキシーがベアメタルホストの Redfish メッセージレジストリーにアクセスできるようになります。

- a. 以下の YAML を **hw-event-bmc-secret.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: redfish-basic-auth
type: Opaque
stringData: 1
  username: <bmc_username>
  password: <bmc_password>
  # BMC host DNS or IP address
  hostaddr: <bmc_host_ip_address>
```

- 1 **stringData** の下に、さまざまな項目のプレーンテキスト値を入力します。

- b. **Secret** CR を作成します。

```
$ oc create -f hw-event-bmc-secret.yaml
```

関連情報

- [ローカルボリュームを使用した永続ストレージ](#)

9.5. ベアメタルイベント REST API リファレンスへのアプリケーションのサブスクリプション

ベアメタルイベント REST API を使用して、親ノードで生成されるベアメタルイベントにアプリケーションをサブスクライブします。

リソースアドレス `/cluster/node/<node_name>/redfish/event` を使用して、アプリケーションを Redfish イベントにサブスクライブします。 `<node_name>` は、アプリケーションを実行するクラスターノードに置き換えます。

cloud-event-consumer アプリケーションコンテナおよび **cloud-event-proxy** サイドカーコンテナを別のアプリケーション Pod にデプロイします。 **cloud-event-consumer** アプリケーションは、アプリケーション Pod の **cloud-event-proxy** コンテナにサブスクライブします。

次の API エンドポイントを使用して、アプリケーション Pod の `http://localhost:8089/api/ocloudNotifications/v1/` にある **cloud-event-proxy** コンテナによって投稿された Redfish イベントに **cloud-event-consumer** アプリケーションをサブスクライブします。

- `/api/ocloudNotifications/v1/subscriptions`
 - **POST**: 新しいサブスクリプションを作成します。
 - **GET**: サブスクリプションの一覧を取得します。
- `/api/ocloudNotifications/v1/subscriptions/<subscription_id>`
 - **PUT**: 指定されたサブスクリプション ID に新しいステータス ping 要求を作成します。
- `/api/ocloudNotifications/v1/health`
 - **GET**: **ocloudNotifications** API の正常性ステータスを返します



注記

9089 は、アプリケーション Pod にデプロイされた **cloud-event-consumer** コンテナのデフォルトポートです。必要に応じて、アプリケーションに異なるポートを設定できます。

api/ocloudNotifications/v1/subscriptions

HTTP メソッド

GET api/ocloudNotifications/v1/subscriptions

説明

サブスクリプションのリストを返します。サブスクリプションが存在する場合は、サブスクリプションの一覧とともに **200 OK** のステータスコードが返されます。

API 応答の例

```
[
  {
    "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "endpointUri": "http://localhost:9089/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
  }
]
```

HTTP メソッド

POST api/ocloudNotifications/v1/subscriptions

説明

新しいサブスクリプションを作成します。サブスクリプションが正常に作成されるか、すでに存在する場合は、**201 Created** ステータスコードが返されます。

表9.1 クエリーパラメーター

パラメーター	型
subscription	data

ペイロードの例

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP メソッド

GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

説明

ID が <subscription_id> のサブスクリプションの詳細を返します。

表9.2 クエリーパラメーター

パラメーター	型
<subscription_id>	string

API 応答の例

```
{
  "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri": "http://localhost:9089/api/ocloudNotifications/v1/dummy",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/health/

HTTP メソッド

GET api/ocloudNotifications/v1/health/

説明

ocloudNotifications REST API の正常性ステータスを返します。

API 応答の例

```
OK
```

9.6. PTP またはベアメタルイベントに HTTP トランスポートを使用するためのコンシューマーアプリケーションの移行

以前に PTP またはベアメタルイベントのコンシューマーアプリケーションをデプロイしている場合は、HTTP メッセージトランスポートを使用するようにアプリケーションを更新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ログインしている。
- PTP Operator または Bare Metal Event Relay を、デフォルトで HTTP トランスポートを使用するバージョン 4.13 以降に更新している。

手順

1. HTTP トランスポートを使用するようにイベントコンシューマーアプリケーションを更新します。クラウドイベントサイドカーデプロイメントの **http-event-publishers** 変数を設定します。
たとえば、PTP イベントが設定されているクラスターでは、以下の YAML スニペットはクラウドイベントサイドカーデプロイメントを示しています。

```
containers:
  - name: cloud-event-sidecar
    image: cloud-event-sidecar
    args:
      - "--metrics-addr=127.0.0.1:9091"
      - "--store-path=/store"
      - "--transport-host=consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043"
      - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043" ❶
      - "--api-port=8089"
```

- ❶ PTP Operator は、PTP イベントを生成するホストに対して **NODE_NAME** を自動的に解決します。**compute-1.example.com** はその例です。

ベアメタルイベントが設定されているクラスターでは、クラウドイベントサイドカーデプロイメント CR で **http-event-publishers** フィールドを **hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043** に設定します。

2. **consumer-events-subscription-service** サービスをイベントコンシューマーアプリケーションと併せてデプロイします。以下に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
name: consumer-events-subscription-service
```

```
namespace: cloud-events
labels:
  app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

第10章 HUGE PAGE の機能およびそれらがアプリケーションによって消費される仕組み

10.1. HUGE PAGE の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 256,000 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トランслーションルックアサイドバッファ (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピングの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

Huge Page とは、4Ki より大きいメモリーページのことです。x86_64 アーキテクチャーでは、2Mi と 1Gi の 2 つが一般的な Huge Page サイズです。別のアーキテクチャーではサイズは異なります。Huge Page を使用するには、アプリケーションが認識できるようにコードを書き込む必要があります。Transparent Huge Pages (THP) は、アプリケーションによる認識なしに、Huge Page の管理を自動化しようとしていますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

OpenShift Container Platform では、Pod のアプリケーションが事前に割り当てられた Huge Page を割り当て、消費することができます。

10.2. HUGE PAGE がアプリケーションによって消費される仕組み

ノードは、Huge Page の容量をレポートできるように Huge Page を事前に割り当てる必要があります。ノードは、単一サイズの Huge Page のみを事前に割り当てることができます。

Huge Page は、リソース名の **hugepages-<size>** を使用してコンテナレベルのリソース要件で消費可能です。この場合、サイズは特定のノードでサポートされる整数値を使用した最もコンパクトなバイナリー表記です。たとえば、ノードが 2048KiB ページサイズをサポートする場合、これはスケジューリング可能なリソース **hugepages-2Mi** を公開します。CPU やメモリーとは異なり、Huge Page はオーバーコミットをサポートしません。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
    privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
```

```

name: hugepage
resources:
  limits:
    hugepages-2Mi: 100Mi ❶
    memory: "1Gi"
    cpu: "1"
  volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

```

- ❶ **hugepages** のメモリー量は、実際に割り当てる量に指定します。この値は、ページサイズで乗算した **hugepages** のメモリー量に指定しないでください。たとえば、Huge Page サイズが 2MB と仮定し、アプリケーションに Huge Page でバックアップする RAM 100 MB を使用する場合には、Huge Page は 50 に指定します。OpenShift Container Platform により、計算処理が実行されます。上記の例にあるように、**100MB** を直接指定できます。

指定されたサイズの Huge Page の割り当て

プラットフォームによっては、複数の Huge Page サイズをサポートするものもあります。特定のサイズの Huge Page を割り当てるには、Huge Page の起動コマンドパラメーターの前に、Huge Page サイズの選択パラメーター **hugepagesz=<size>** を指定してください。**<size>** の値は、バイトで指定する必要があります。その際、オプションでスケール接尾辞 [**kKmMgG**] を指定できます。デフォルトの Huge Page サイズは、**default_hugepagesz=<size>** の起動パラメーターで定義できます。

Huge page の要件

- Huge Page 要求は制限と同じでなければなりません。制限が指定されているにもかかわらず、要求が指定されていない場合には、これがデフォルトになります。
- Huge Page は、Pod のスコープで分割されます。コンテナの分割は、今後のバージョンで予定されています。
- Huge Page がサポートする **EmptyDir** ボリュームは、Pod 要求よりも多くの Huge Page メモリーを消費することはできません。
- **shmget()** で **SHM_HUGETLB** を使用して Huge Page を消費するアプリケーションは、**proc/sys/vm/hugetlb_shm_group** に一致する補助グループで実行する必要があります。

10.3. DOWNWARD API を使用した HUGE PAGE リソースの使用

Downward API を使用して、コンテナで使用する Huge Page リソースに関する情報を挿入できます。

リソースの割り当ては、環境変数、ボリュームプラグイン、またはその両方として挿入できます。コンテナで開発および実行するアプリケーションは、指定されたボリューム内の環境変数またはファイルを読み取ることで、利用可能なリソースを判別できます。

手順

1. 以下の例のような **hugepages-volume-pod.yaml** ファイルを作成します。

```

apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-

```

```

labels:
  app: hugepages-example
spec:
  containers:
  - securityContext:
      capabilities:
        add: [ "IPC_LOCK" ]
      image: rhel7:latest
      command:
      - sleep
      - inf
      name: example
      volumeMounts:
      - mountPath: /dev/hugepages
        name: hugepage
      - mountPath: /etc/podinfo
        name: podinfo
      resources:
        limits:
          hugepages-1Gi: 2Gi
          memory: "1Gi"
          cpu: "1"
        requests:
          hugepages-1Gi: 2Gi
      env:
      - name: REQUESTS_HUGEPAGES_1GI <.>
        valueFrom:
          resourceFieldRef:
            containerName: example
            resource: requests.hugepages-1Gi
      volumes:
      - name: hugepage
        emptyDir:
          medium: HugePages
      - name: podinfo
        downwardAPI:
          items:
          - path: "hugepages_1G_request" <.>
            resourceFieldRef:
              containerName: example
              resource: requests.hugepages-1Gi
            divisor: 1Gi

```

<.> では、**requests.hugepages-1Gi** からリソースの使用を読み取り、**REQUESTS_HUGEPAGES_1GI** 環境変数としてその値を公開するように指定し、2つ目の<.>は、**requests.hugepages-1Gi** からのリソースの使用を読み取り、**/etc/podinfo/hugepages_1G_request** ファイルとして値を公開するように指定します。

2. **hugepages-volume-pod.yaml** ファイルから Pod を作成します。

```
$ oc create -f hugepages-volume-pod.yaml
```

検証

1. **REQUESTS_HUGEPAGES_1GI** 環境変数の値を確認します。

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
  -- env | grep REQUESTS_HUGEPAGES_1Gi
```

出力例

```
REQUESTS_HUGEPAGES_1Gi=2147483648
```

2. `/etc/podinfo/hugepages_1G_request` ファイルの値を確認します。

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
  -- cat /etc/podinfo/hugepages_1G_request
```

出力例

```
2
```

関連情報

- [コンテナによる Downward API オブジェクト使用の許可](#)

10.4. 起動時の HUGE PAGE 設定

ノードは、OpenShift Container Platform クラスタで使用される Huge Page を事前に割り当てる必要があります。Huge Page を予約する方法は、ブート時とランタイム時に実行する2つの方法があります。ブート時の予約は、メモリーが大幅に断片化されていないために成功する可能性が高くなります。Node Tuning Operator は、現時点で特定のノードでの Huge Page のブート時の割り当てをサポートします。

手順

ノードの再起動を最小限にするには、以下の手順の順序に従う必要があります。

1. ラベルを使用して同じ Huge Page 設定を必要とするすべてのノードにラベルを付けます。

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. 以下の内容でファイルを作成し、これに `hugepages-tuned-boottime.yaml` という名前を付けます。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
    [main]
    summary=Boot time configuration for hugepages
    include=openshift-node
    [bootloader]
```

```

cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 3
name: openshift-node-hugepages

recommend:
- machineConfigLabels: 4
  machineconfiguration.openshift.io/role: "worker-hp"
  priority: 30
  profile: openshift-node-hugepages

```

- 1** チューニングされたリソースの **name** を **hugepages** に設定します。
- 2** Huge Page を割り当てる **profile** セクションを設定します。
- 3** 一部のプラットフォームではさまざまなサイズの Huge Page をサポートするため、パラメーターの順序に注意してください。
- 4** マシン設定プールベースのマッチングを有効にします。

3. チューニングされた **hugepages** オブジェクトの作成

```
$ oc create -f hugepages-tuned-boottime.yaml
```

4. 以下の内容でファイルを作成し、これに **hugepages-mcp.yaml** という名前を付けます。

```

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

```

5. マシン設定プールを作成します。

```
$ oc create -f hugepages-mcp.yaml
```

断片化されていないメモリが十分にある場合、**worker-hp** マシン設定プールのすべてのノードには 50 2Mi の Huge Page が割り当てられているはずです。

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```



注記

TuneD ブートローダープラグインは、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみサポートします。

10.5. TRANSPARENT HUGE PAGES (THP) の無効化

Transparent Huge Page (THP) は、Huge Page を作成し、管理し、使用するためのほとんどの要素を自動化しようとします。THP は Huge Page を自動的に管理するため、すべてのタイプのワークロードに対して常に最適に処理される訳ではありません。THP は、多くのアプリケーションが独自の Huge Page を処理するため、パフォーマンス低下につながる可能性があります。したがって、THP を無効にすることを検討してください。以下の手順では、Node Tuning Operator (NTO) を使用して THP を無効にする方法を説明します。

手順

1. 以下の内容でファイルを作成し、**thp-disable-tuned.yaml** という名前を付けます。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: thp-workers-profile
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
    [main]
    summary=Custom tuned profile for OpenShift to turn off THP on worker nodes
    include=openshift-node

    [vm]
    transparent_hugepages=never
    name: openshift-thp-never-worker

  recommend:
  - match:
    - label: node-role.kubernetes.io/worker
    priority: 25
    profile: openshift-thp-never-worker
```

2. Tuned オブジェクトを作成します。

```
$ oc create -f thp-disable-tuned.yaml
```

3. アクティブなプロファイルのリストを確認します。

```
$ oc get profile -n openshift-cluster-node-tuning-operator
```

検証

- ノードのいずれかにログインし、通常の THP チェックを実行して、ノードがプロファイルを正常に適用したかどうかを確認します。

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

出力例

```
always madvise [never]
```

第11章 低遅延チューニング

11.1. 低レイテンシーについて

Telco / 5G の領域でのエッジコンピューティングの台頭は、レイテンシーと輻輳を軽減し、アプリケーションのパフォーマンスを向上させる上で重要な役割を果たします。

簡単に言うと、レイテンシーは、データ (パケット) が送信側から受信側に移動し、受信側の処理後に送信側に戻る速度を決定します。レイテンシーによる遅延を最小限に抑えた状態でネットワークアーキテクチャーを維持することが 5G のネットワークパフォーマンス要件を満たすのに鍵となります。4G テクノロジーと比較し、平均レイテンシーが 50 ms の 5G では、レイテンシーの数値を 1ms 以下にするようにターゲットが設定されます。このレイテンシーの減少により、ワイヤレスのスループットが 10 倍向上します。

Telco 領域にデプロイされるアプリケーションの多くは、ゼロパケットロスに耐えられる低レイテンシーを必要とします。パケットロスをゼロに調整すると、ネットワークのパフォーマンス低下させる固有の問題を軽減することができます。詳細は、[Tuning for Zero Packet Loss in Red Hat OpenStack Platform \(RHOSP\)](#) を参照してください。

エッジコンピューティングの取り組みは、レイテンシーの削減にも役立ちます。クラウドの端にあり、ユーザーに近いと考えてください。これにより、ユーザーと離れた場所にあるデータセンター間の距離が大幅に削減されるため、アプリケーションの応答時間とパフォーマンスのレイテンシーが短縮されます。

管理者は、すべてのデプロイメントを可能な限り低い管理コストで実行できるように、多数のエッジサイトおよびローカルサービスを一元管理できるようにする必要があります。また、リアルタイムの低レイテンシーおよび高パフォーマンスを実現するために、クラスターの特定のノードをデプロイし、設定するための簡単な方法も必要になります。低レイテンシーノードは、Cloud-native Network Functions (CNF) や Data Plane Development Kit (DPDK) などのアプリケーションに役立ちます。

現時点で、OpenShift Container Platform はリアルタイムの実行および低レイテンシーを実現するために OpenShift Container Platform クラスターでソフトウェアを調整するメカニズムを提供します (約 20 マイクロ秒未満の応答時間)。これには、カーネルおよび OpenShift Container Platform の設定値のチューニング、カーネルのインストール、およびマシンの再設定が含まれます。ただし、この方法では 4 つの異なる Operator を設定し、手動で実行する場合に複雑であり、間違いが生じる可能性がある多くの設定を行う必要があります。

OpenShift Container Platform は、ノードチューニング Operator を使用して自動チューニングを実装し、OpenShift Container Platform 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ます。クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更をより容易に実行することができます。管理者は、カーネルを kernel-rt に更新するかどうかを指定し、Pod の infra コンテナなどのクラスターおよびオペレーティングシステムのハウスキーピング向けに CPU を予約して、アプリケーションコンテナがワークロードを実行するように CPU を分離することができます。



注記

現在、CPU 負荷分散の無効化は cgroup v2 ではサポートされていません。その結果、cgroup v2 が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、cgroup v2 を有効にすることは推奨されません。

OpenShift Container Platform は、さまざまな業界環境の要求を満たすように **PerformanceProfile** を調整できる Node Tuning Operator のワークロードヒントもサポートします。ワークロードのヒントは、**highPowerConsumption** (消費電力が増加する代わりにレイテンシーを非常に低く抑える) と

realTime (最適なレイテンシーを優先) で利用できます。これらのヒントの **true/false** 設定の組み合わせを使用して、アプリケーション固有のワークロードプロファイルと要件を処理できます。

ワークロードのヒントは、業界セクターの設定に対するパフォーマンスの微調整を簡素化します。1つのサイズですべてに対応するアプローチの代わりに、ワークロードのヒントは、以下を優先するなどの使用パターンに対応できます。

- 低レイテンシー
- リアルタイム機能
- 電力の効率的な使用

理想的な世界では、これらすべてが優先されます。実際の生活では、他の人を犠牲にしてやってくる人もいます。Node Tuning Operator は、ワークロードの期待を認識し、ワークロードの要求をより適切に満たすことができるようになりました。クラスター管理者は、ワークロードがどのユースケースに分類されるかを指定できるようになりました。Node Tuning Operator は、**PerformanceProfile** を使用して、ワークロードのパフォーマンス設定を微調整します。

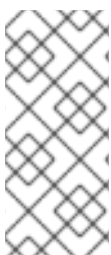
アプリケーションが動作している環境は、その動作に影響を与えます。厳密なレイテンシー要件のない一般的なデータセンターの場合、一部の高性能ワークロード Pod の CPU パーティショニングを可能にする最小限のデフォルトチューニングのみが必要です。レイテンシーが優先されるデータセンターやワークロードの場合でも、消費電力を最適化するための対策が講じられています。最も複雑なケースは、製造機械やソフトウェア無線などのレイテンシーの影響を受けやすい機器に近いクラスターです。この最後のクラスのデプロイメントは、多くの場合、ファアーエッジと呼ばれます。ファアーエッジデプロイメントの場合、超低レイテンシーが最優先事項であり、電力管理を犠牲にして実現されます。

OpenShift Container Platform バージョン 4.10 およびそれ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、低レイテンシーのパフォーマンスを実現しました。現在、この機能はノードチューニング Operator の一部です。

11.1.1. 低レイテンシーおよびリアルタイムのアプリケーションのハイパースレッディングについて

ハイパースレッディングは、物理 CPU プロセッサコアが2つの論理コアとして機能することを可能にする Intel プロセッサテクノロジーで、2つの独立したスレッドを同時に実行します。ハイパースレッディングにより、並列処理が効果的な特定のワークロードタイプのシステムスループットを向上できます。デフォルトの OpenShift Container Platform 設定では、ハイパースレッディングがデフォルトで有効にされることが予想されます。

通信アプリケーションの場合、可能な限りレイテンシーを最小限に抑えられるようにアプリケーションインフラストラクチャーを設計することが重要です。ハイパースレッディングは、パフォーマンスを低下させる可能性があり、低レイテンシーを必要とするコンピュート集約型のワークロードのスループットにマイナスの影響を及ぼす可能性があります。ハイパースレッディングを無効にすると、予測可能なパフォーマンスが確保され、これらのワークロードの処理時間が短縮されます。



注記

ハイパースレッディングの実装および設定は、OpenShift Container Platform を実行しているハードウェアによって異なります。ハードウェアに固有のハイパースレッディング実装についての詳細は、関連するホストハードウェアのチューニング情報を参照してください。ハイパースレッディングを無効にすると、クラスターのコアごとにコストが増大する可能性があります。

関連情報

- クラスターのハイパースレッディングの設定

11.2. リアルタイムおよび低レイテンシーワークロードのプロビジョニング

多くの企業や組織は、非常に高性能なコンピューティングを必要としており、とくに金融業界や通信業界では、低い、予測可能なレイテンシーが必要になる場合があります。こうした業界特有の要件に対して、OpenShift Container Platform では、OpenShift Container Platform アプリケーションの低遅延性能と一貫した応答速度を実現するための自動チューニングを実施する Node Tuning Operator を提供しています。

クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更を加えることができます。管理者は、カーネルを kernel-rt (リアルタイム) に更新するか、Pod infra コンテナを含むクラスターと OS のハウスキーピング業務用に CPU を確保するか、アプリケーションコンテナ用に CPU を分離してワークロードを実行するか、未使用 CPU を無効にして電力消費を抑えるかを指定することができます。



警告

保証された CPU を必要とするアプリケーションと組み合わせて実行プローブを使用すると、レイテンシースパイクが発生する可能性があります。代わりに、適切に設定されたネットワークプローブのセットなど、他のプローブを使用することを推奨します。



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、これらの機能は Node Tuning Operator の一部です。

11.2.1. リアルタイムの既知の制限



注記

ほとんどのデプロイメントで、3つのコントロールプレーンノードと3つのワーカーノードを持つ標準クラスターを使用する場合、kernel-rt はワーカーノードでのみサポートされます。OpenShift Container Platform デプロイメントのコンパクトノードと単一ノードには例外があります。単一ノードへのインストールの場合、kernel-rt は単一のコントロールプレーンノードでサポートされます。

リアルタイムモードを完全に使用するには、コンテナを昇格した権限で実行する必要があります。権限の付与についての情報は、[Set capabilities for a Container](#) を参照してください。

OpenShift Container Platform は許可される機能を制限するため、**SecurityContext** を作成する必要があります。ある場合もあります。



注記

この手順は、Red Hat Enterprise Linux CoreOS (RHCOS) システムを使用したベアメタルのインストールで完全にサポートされます。

パフォーマンスの期待値を設定する必要があるということは、リアルタイムカーネルがあらゆる問題の解決策ではないということを意味します。リアルタイムカーネルは、一貫性のある、低レイテンシーの、決定論に基づく予測可能な応答時間を提供します。リアルタイムカーネルに関連して、追加のカーネルオーバーヘッドがあります。これは、主に個別にスケジュールされたスレッドでハードウェア割り込みを処理することによって生じます。一部のワークロードのオーバーヘッドが増加すると、スループット全体が低下します。ワークロードによって異なりますが、パフォーマンスの低下の程度は0%から30%の範囲になります。ただし、このコストは決定論をベースとしています。

11.2.2. リアルタイム機能のあるワーカーのプロビジョニング

1. オプション: ノードを OpenShift Container Platform クラスターに追加します。 [システムチューニング用の BIOS パラメーターの設定](#) を参照してください。
2. `oc` コマンドを使用して、リアルタイム機能を必要とするワーカーノードにラベル `worker-rt` を追加します。
3. リアルタイムノード用の新しいマシン設定プールを作成します。

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

マシン設定プール `worker-rt` は、`worker-rt` というラベルを持つノードのグループに対して作成されることに注意してください。

4. ノードロールラベルを使用して、ノードを適切なマシン設定プールに追加します。



注記

リアルタイムワークロードで設定するノードを決定する必要があります。クラスター内のすべてのノード、またはノードのサブセットを設定できます。すべてのノードが専用のマシン設定プールの一部であることを期待する Node Tuning Operator。すべてのノードを使用する場合は、Node Tuning Operator がワーカーノードのロールラベルを指すようにする必要があります。サブセットを使用する場合、ノードを新規のマシン設定プールにグループ化する必要があります。

- ハウスキーピングコアの適切なセットと **realTimeKernel: enabled: true** を設定して **PerformanceProfile** を作成します。
- PerformanceProfile** で **machineConfigPoolSelector** を設定する必要があります:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  ...
  realTimeKernel:
    enabled: true
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
  machineConfigPoolSelector:
    machineconfiguration.openshift.io/role: worker-rt
```

- 一致するマシン設定プールがラベルを持つことを確認します。

```
$ oc describe mcp/worker-rt
```

出力例

```
Name:      worker-rt
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker-rt
```

- OpenShift Container Platform はノードの設定を開始しますが、これにより複数の再起動が伴う可能性があります。ノードが起動し、安定するのを待機します。特定のハードウェアの場合に、これには長い時間がかかる可能性があります、ノードごとに 20 分の時間がかかることが予想されます。
- すべてが予想通りに機能していることを確認します。

11.2.3. リアルタイムカーネルのインストールの確認

以下のコマンドを使用して、リアルタイムカーネルがインストールされていることを確認します。

```
$ oc get node -o wide
```

4.18.0-305.30.1.rt7.102.el8_4.x86_64 cri-o://1.26.0-99.rhaos4.10.gitc3131de.el8 の文字列を含むロール **worker-rt** を持つワーカーに注意してください。

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
EXTERNAL-IP	OS-IMAGE			KERNEL-VERSION	
CONTAINER-RUNTIME					
rt-worker-0.example.com	Ready	worker,worker-rt	5d17h	v1.26.0	
128.66.135.107	<none>	Red Hat Enterprise Linux	CoreOS	46.82.202008252340-0	(Ootpa)
4.18.0-305.30.1.rt7.102.el8_4.x86_64		cri-o://1.26.0-99.rhaos4.10.gitc3131de.el8			
[...]					

11.2.4. リアルタイムで機能するワークロードの作成

リアルタイム機能を使用するワークロードを準備するには、以下の手順を使用します。

手順

1. QoS クラスの **Guaranteed** を指定して Pod を作成します。
2. オプション: DPDK の CPU 負荷分散を無効にします。
3. 適切なノードセクターを割り当てます。

アプリケーションを作成する場合には、[アプリケーションのチューニングとデプロイメント](#) に記載されている一般的な推奨事項に従ってください。

11.2.5. QoS クラスの **Guaranteed** を指定した Pod の作成

QoS クラスの **Guaranteed** が指定されている Pod を作成する際には、以下を考慮してください。

- Pod のすべてのコンテナにはメモリー制限およびメモリー要求があり、それらは同じである必要があります。
- Pod のすべてのコンテナには CPU の制限と CPU 要求が必要であり、それらは同じである必要があります。

以下の例は、1つのコンテナを持つ Pod の設定ファイルを示しています。コンテナにはメモリー制限とメモリー要求があり、どちらも 200 MiB に相当します。コンテナには CPU 制限と CPU 要求があり、どちらも 1CPU に相当します。

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: <image-pull-spec>
    resources:
      limits:
        memory: "200Mi"
        cpu: "1"
      requests:
        memory: "200Mi"
        cpu: "1"
```

1. Pod を作成します。

```
$ oc apply -f qos-pod.yaml --namespace=qos-example
```

2. Pod についての詳細情報を表示します。

```
$ oc get pod qos-demo --namespace=qos-example --output=yaml
```

出力例

```
spec:
  containers:
    ...
status:
  qosClass: Guaranteed
```



注記

コンテナが独自のメモリー制限を指定するものの、メモリー要求を指定しない場合、OpenShift Container Platform は制限に一致するメモリー要求を自動的に割り当てます。同様に、コンテナが独自の CPU 制限を指定するものの、CPU 要求を指定しない場合、OpenShift Container Platform は制限に一致する CPU 要求を自動的に割り当てます。

11.2.6. オプション: DPDK 用の CPU 負荷分散の無効化

CPU 負荷分散を無効または有効にする機能は CRI-O レベルで実装されます。CRI-O のコードは、以下の要件を満たす場合にのみ CPU の負荷分散を無効または有効にします。

- Pod は **performance-`<profile-name>`** ランタイムクラスを使用する必要があります。以下に示すように、パフォーマンスプロファイルのステータスを確認して、適切な名前を取得できます。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
...
status:
  ...
runtimeClass: performance-manual
```



注記

現在、cgroup v2 では CPU 負荷分散の無効化はサポートされていません。

Node Tuning Operator は、関連ノード下での高性能ランタイムハンドラー config snippet の作成と、クラスター下での高性能ランタイムクラスの作成を担当します。これには、CPU 負荷分散の設定機能を有効にすることを除くと、デフォルトのランタイムハンドラーと同じ内容が含まれます。

Pod の CPU 負荷分散を無効にするには、**Pod** 仕様に以下のフィールドが含まれる必要があります。

```
apiVersion: v1
kind: Pod
metadata:
  ...
annotations:
  ...
  cpu-load-balancing.crio.io: "disable"
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```




注記

CPU マネージャーの静的ポリシーが有効にされている場合に、CPU 全体を使用する Guaranteed QoS を持つ Pod について CPU 負荷分散を無効にします。これ以外の場合に CPU 負荷分散を無効にすると、クラスター内の他のコンテナのパフォーマンスに影響する可能性があります。

11.2.7. 適切なノードセクターの割り当て

Pod をノードに割り当てる方法として、以下に示すようにパフォーマンスプロファイルが使用するものと同じノードセクターを使用することが推奨されます。

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  # ...
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
```

ノードセクターの詳細は、[Placing pods on specific nodes using node selectors](#) を参照してください。

11.2.8. リアルタイム機能を備えたワーカーへのワークロードのスケジューリング

Node Tuning Operator によって低レイテンシー用に設定されたマシン設定プールに接続されているノードに一致するラベルセクターを使用します。詳細は、[Assigning pods to nodes](#) を参照してください。

11.2.9. CPU をオフラインにすることで消費電力を削減

一般に、通信のワークロードを予測できます。すべての CPU リソースが必要なわけではない場合、Node Tuning Operator を使用すると、未使用の CPU をオフラインにして、パフォーマンスプロファイルを手動で更新することにより、消費電力を削減できます。

未使用の CPU をオフラインにするには、次のタスクを実行する必要があります。

1. パフォーマンスプロファイルでオフライン CPU を設定し、YAML ファイルの内容を保存します。

オフライン CPU を使用したパフォーマンスプロファイルの例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - intel_idle.max_cstate=0
    - idle=poll
```

```

cpu:
  isolated: "2-23,26-47"
  reserved: "0,1,24,25"
  offlined: "48-59" ❶
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
numa:
  topologyPolicy: single-numa-node
realTimeKernel:
  enabled: true

```

- ❶ オプション: **offlined** フィールドに CPU をリストして、指定した CPU をオフラインにすることができます。

2. 次のコマンドを実行して、更新されたプロファイルを適用します。

```
$ oc apply -f my-performance-profile.yaml
```

11.2.10. オプション: 省電力設定

優先度の高いワークロードのレイテンシーやスループットに影響を与えることなく、優先度の高いワークロードと同じ場所にある優先度の低いワークロードを持つノードの省電力を有効にすることができます。ワークロード自体を変更することなく、省電力が可能です。



重要

この機能は、Intel Ice Lake 以降の世代の Intel CPU でサポートされています。プロセッサの機能は、優先度の高いワークロードのレイテンシーとスループットに影響を与える可能性があります。

省電力設定でノードを設定するときは、優先度の高いワークロードを Pod レベルのパフォーマンス設定で設定する必要があります。つまり、Pod で使用されるすべてのコアにその設定が適用されます。

Pod レベルで P ステートと C ステートを無効にすることで、優先度の高いワークロードを設定して、最高のパフォーマンスと最小の待機時間を実現できます。

表11.1 優先度の高いワークロードの設定

アノテーション	説明
<pre> annotations: cpu-c-states.crio.io: "disable" cpu-freq-governor.crio.io: "<governor>" </pre>	<p>C ステートを無効にし、CPU スケーリングのガバナータイプを指定することで、Pod に最高のパフォーマンスを提供します。performance ガバナーは、優先度の高いワークロードに推奨されます。</p>

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た

1. **per-pod-power-management** を **true** に設定して **PerformanceProfile** を生成します。

```
$ podman run --entrypoint performance-profile-creator -v \
/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-
operator:v4.13 \
--mcp-name=worker-cnf --reserved-cpu-count=20 --rt-kernel=true \
--split-reserved-cpus-across-numa=false --topology-manager-policy=single-numa-node \
--must-gather-dir-path /must-gather -power-consumption-mode=low-latency \ 1
--per-pod-power-management=true > my-performance-profile.yaml
```

- 1 **per-pod-power-management** が **true** に設定されている場合、**power-consumption-mode** は **default** または **low-latency** である必要があります。

perPodPowerManagement を使用した PerformanceProfile の例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  [...]
workloadHints:
  realTime: true
  highPowerConsumption: false
  perPodPowerManagement: true
```

2. デフォルトの **cpufreq** ガバナーを、**PerformanceProfile** カスタムリソース (CR) で追加のカーネル引数として設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  ...
  additionalKernelArgs:
  - cpufreq.default_governor=schedutil 1
```

- 1 **schedutil** ガバナーの使用が推奨されますが、**ondemand** ガバナーや **powersave** ガバナーなどの他のガバナーを使用することもできます。

3. **TunedPerformancePatch** CR で最大 CPU 周波数を設定します。

```
spec:
  profile:
  - 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

4. 必要なアノテーションを優先度の高いワークロード Pod に追加します。注釈は **default** 設定を上書きします。

優先度の高いワークロードアノテーションの例

```

apiVersion: v1
kind: Pod
metadata:
  ...
  annotations:
    ...
    cpu-c-states.crio.io: "disable"
    cpu-freq-governor.crio.io: "<governor>"
    ...
  spec:
    ...
    runtimeClassName: performance-<profile_name>
    ...

```

5. Pod を再起動します。

関連情報

- 推奨されるファームウェア設定の詳細は、[vDU クラスターホストの推奨されるファームウェア設定](#) を参照してください。

11.2.11. Guaranteed Pod の分離された CPU のデバイス割り込み処理の管理

Node Tuning Operator は、ホスト CPU を、Pod Infra コンテナを含むクラスターとオペレーティングシステムのハウスキーピング業務用の予約 CPU と、ワークロードを実行するアプリケーションコンテナ用の分離 CPU に分割して管理することができます。これにより、低レイテンシーのワークロード用の CPU を isolated (分離された CPU) として設定できます。

デバイスの割り込みについては、Guaranteed Pod が実行されている CPU を除き、CPU のオーバーロードを防ぐためにすべての分離された CPU および予約された CPU 間で負荷が分散されます。Guaranteed Pod の CPU は、関連するアノテーションが Pod に設定されている場合にデバイス割り込みを処理できなくなります。

パフォーマンスプロファイルで、**globallyDisableIrqLoadBalancing** は、デバイス割り込みが処理されるかどうかを管理するために使用されます。特定のワークロードでは、予約された CPU は、デバイスの割り込みを処理するのに常に十分な訳ではないため、デバイスの割り込みは分離された CPU でグローバルに無効化されていません。デフォルトでは、Node Tuning Operator は分離された CPU でのデバイス割り込みを無効にしません。

ワークロードの低レイテンシーを確保するには、一部の (すべてではない) Pod で、それらが実行されている CPU がデバイス割り込みを処理しないようにする必要があります。Pod アノテーション **irq-load-balancing.crio.io** は、デバイス割り込みが処理されるかどうかを定義するために使用されます。CRI-O は (設定されている場合)、Pod が実行されている場合にのみデバイス割り込みを無効にします。

11.2.11.1. CPU CFS クォータの無効化

保証された個々の Pod の CPU スロットル調整を減らすには、アノテーション **cpu-quota.crio.io: "disable"** を付けて、Pod 仕様を作成します。このアノテーションは、Pod の実行時に CPU Completely Fair Scheduler (CFS) のクォータを無効にします。次の Pod 仕様には、このアノテーションが含まれています。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cpu-quota.crio.io: "disable"
spec:
  runtimeClassName: performance-<profile_name>
...
```



注記

CPU マネージャーの静的ポリシーが有効になっている場合、および CPU 全体を使用する Guaranteed QoS を持つ Pod の場合にのみ、CPU CFS クォータを無効にします。これ以外の場合に CPU CFS クォータを無効にすると、クラスター内の他のコンテナのパフォーマンスに影響を与える可能性があります。

11.2.11.2. Node Tuning Operator でのグローバルデバイス割り込み処理の無効化

分離された CPU セットのグローバルデバイス割り込みを無効にするように Node Tuning Operator を設定するには、パフォーマンスプロファイルの **globallyDisableIrqLoadBalancing** フィールドを **true** に設定します。**true** の場合、競合する Pod アノテーションは無視されます。**false** の場合、すべての CPU 間で IRQ 負荷が分散されます。

パフォーマンスプロファイルのスニペットは、この設定を示しています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  globallyDisableIrqLoadBalancing: true
...
```

11.2.11.3. 個別の Pod の割り込み処理の無効化

個別の Pod の割り込み処理を無効にするには、パフォーマンスプロファイルで **globalDisableIrqLoadBalancing** が **false** に設定されていることを確認します。次に、Pod 仕様で、**irq-load-balancing.crio.io** Pod アノテーションを **disable** に設定します。次の Pod 仕様には、このアノテーションが含まれています。

```
apiVersion: performance.openshift.io/v2
kind: Pod
metadata:
  annotations:
    irq-load-balancing.crio.io: "disable"
spec:
  runtimeClassName: performance-<profile_name>
...
```

11.2.12. デバイス割り込み処理を使用するためのパフォーマンスプロファイルのアップグレード

Node Tuning Operator パフォーマンスプロファイルのカスタムリソース定義 (CRD) を v1 または v1alpha1 から v2 にアップグレードする場合、**globallyDisableIrqLoadBalancing** は **true** に設定されます。



注記

globallyDisableIrqLoadBalancing は、IRQ ロードバランシングを分離 CPU セットに対して無効にするかどうかを切り替えます。このオプションを **true** に設定すると、分離 CPU セットの IRQ ロードバランシングが無効になります。オプションを **false** に設定すると、IRQ をすべての CPU 間でバランスさせることができます。

11.2.12.1. サポート対象の API バージョン

Node Tuning Operator は、パフォーマンスプロファイル **apiVersion** フィールドの **v2**、**v1**、および **v1alpha1** をサポートします。v1 および v1alpha1 API は同一です。v2 API には、デフォルト値の **false** が設定されたオプションのブール値フィールド **globallyDisableIrqLoadBalancing** が含まれます。

11.2.12.1.1. Node Tuning Operator API の v1alpha1 から v1 へのアップグレード

Node Tuning Operator API バージョンを v1alpha1 から v1 にアップグレードする場合、v1alpha1 パフォーマンスプロファイルは None 変換ストラテジーを使用してオンザフライで変換され、API バージョン v1 の Node Tuning Operator に提供されます。

11.2.12.1.2. Node Tuning Operator API の v1alpha1 または v1 から v2 へのアップグレード

古い Node Tuning Operator API バージョンからアップグレードする場合、既存の v1 および v1alpha1 パフォーマンスプロファイルは、**globallyDisableIrqLoadBalancing** フィールドに **true** の値を挿入する変換 Webhook を使用して変換されます。

11.3. パフォーマンスプロファイルによる低レイテンシーを実現するためのノードのチューニング

パフォーマンスプロファイルを使用すると、特定のマシン設定プールに属するノードのレイテンシーの調整を制御できます。設定を指定すると、**PerformanceProfile** オブジェクトは実際のノードレベルのチューニングを実行する複数のオブジェクトにコンパイルされます。

- ノードを操作する **MachineConfig** ファイル。
- Topology Manager、CPU マネージャー、および OpenShift Container Platform ノードを設定する **KubeletConfig** ファイル。
- Node Tuning Operator を設定する Tuned プロファイル。

パフォーマンスプロファイルを使用して、カーネルを kernel-rt に更新して Huge Page を割り当て、ハウスキーピングデータの実行やワークロードの実行用に CPU をパーティションに分割するかどうかを指定できます。



注記

PerformanceProfile オブジェクトを手動で作成するか、Performance Profile Creator (PPC) を使用してパフォーマンスプロファイルを生成することができます。PPC の詳細については、以下の関連情報を参照してください。

パフォーマンスプロファイルの例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "4-15" ①
    reserved: "0-3" ②
  hugepages:
    defaultHugepagesSize: "1G"
  pages:
    - size: "1G"
      count: 16
      node: 0
  realTimeKernel:
    enabled: true ③
  numa: ④
    topologyPolicy: "best-effort"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: "" ⑤
```

- ① このフィールドでは、特定の CPU を分離し、ワークロード用に、アプリケーションコンテナで使用します。ハイパースレッディングが有効な場合に Pod がエラーなしで実行できるようにするには、分離された CPU の数を偶数に設定します。
- ② このフィールドでは、特定の CPU を予約し、ハウスキーピング用に infra コンテナで使用します。
- ③ このフィールドでは、ノード上にリアルタイムカーネルをインストールします。有効な値は **true** または **false** です。**true** 値を設定すると、ノード上にリアルタイムカーネルがインストールされます。
- ④ Topology Manager ポリシーを設定するには、このフィールドを使用します。有効な値は **none** (デフォルト)、**best-effort**、**restricted**、および **single-numa-node** です。詳細は、[Topology Manager Policies](#) を参照してください。
- ⑤ このフィールドを使用してノードセレクターを指定し、パフォーマンスプロファイルを特定のノードに適用します。

関連情報

- Performance Profile Creator (PPC) を使用してパフォーマンスプロファイルを生成する方法の詳細は、[Creating a performance profile](#) を参照してください。

11.3.1. Huge Page の設定

ノードは、OpenShift Container Platform クラスターで使用される Huge Page を事前に割り当てる必要があります。Node Tuning Operator を使用し、特定のノードで Huge Page を割り当てます。

OpenShift Container Platform は、Huge Page を作成し、割り当てる方法を提供します。Node Tuning Operator は、パフォーマンスプロファイルを使用して、これをより簡単に行う方法を提供します。

たとえば、パフォーマンスプロファイルの **hugepages pages** セクションで、**size**、**count**、およびオプションで **node** の複数のブロックを指定できます。

```
hugepages:
  defaultHugepagesSize: "1G"
  pages:
    - size: "1G"
      count: 4
      node: 0 1
```

- 1 **node** は、Huge Page が割り当てられる NUMA ノードです。 **node** を省略すると、ページはすべての NUMA ノード間で均等に分散されます。



注記

更新が完了したことを示す関連するマシン設定プールのステータスを待機します。

これらは、Huge Page を割り当てるのに必要な唯一の設定手順です。

検証

- 設定を確認するには、ノード上の **/proc/meminfo** ファイルを参照します。

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

```
# grep -i huge /proc/meminfo
```

出力例

```
AnonHugePages: ##### ##
ShmemHugePages: 0 kB
HugePages_Total: 2
HugePages_Free: 2
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: ##### ##
Hugetlb: ##### ##
```

- 新規サイズを報告するには、**oc describe** を使用します。

```
$ oc describe node worker-0.ocp4poc.example.com | grep -i huge
```

出力例


```

                hugepages-1g=true
hugepages-###: ###
hugepages-###: ###

```

11.3.2. 複数の Huge Page サイズの割り当て

同じコンテナで異なるサイズの Huge Page を要求できます。これにより、Huge Page サイズの異なる複数のコンテナで設定されるより複雑な Pod を定義できます。

たとえば、サイズ **1G** と **2M** を定義でき、Node Tuning Operator は以下に示すようにノード上に両方のサイズを設定します。

```

spec:
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 1024
      node: 0
      size: 2M
    - count: 4
      node: 1
      size: 1G

```

11.3.3. IRQ 動的負荷分散用ノードの設定

どのコアがデバイス割り込み要求 (IRQ) を受信できるかを制御するために、IRQ 動的負荷分散用にクラスターノードを設定します。

前提条件

- コアを分離するには、すべてのサーバーハードウェアコンポーネントが IRQ アフィニティーをサポートしている必要があります。サーバーのハードウェアコンポーネントが IRQ アフィニティーをサポートしているかどうかを確認するには、サーバーのハードウェア仕様を参照するか、ハードウェアプロバイダーにお問い合わせください。

手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。
2. パフォーマンスプロファイルの **apiVersion** を **performance.openshift.io/v2** を使用するように設定します。
3. **globallyDisableIrqLoadBalancing** フィールドを削除するか、これを **false** に設定します。
4. 適切な分離された CPU と予約された CPU を設定します。以下のスニペットは、2つの CPU を確保するプロファイルを示しています。IRQ 負荷分散は、**isolated** CPU セットで実行されている Pod について有効にされます。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: dynamic-irq-profile
spec:

```

```
cpu:
  isolated: 2-5
  reserved: 0-1
...
```



注記

予約および分離された CPU を設定する場合に、Pod 内の infra コンテナは予約された CPU を使用し、アプリケーションコンテナは分離された CPU を使用します。

5. 排他的な CPU を使用する Pod を作成し、**irq-load-balancing.crio.io** および **cpu-quota.crio.io** アノテーションを **disable** に設定します。以下に例を示します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dynamic-irq-pod
  annotations:
    irq-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
spec:
  containers:
  - name: dynamic-irq-pod
    image: "registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13"
    command: ["sleep", "10h"]
    resources:
      requests:
        cpu: 2
        memory: "200M"
      limits:
        cpu: 2
        memory: "200M"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  runtimeClassName: performance-dynamic-irq-profile
...
```

6. performance-`<profile_name>` の形式で Pod **runtimeClassName** を入力します。ここで、`<profile_name>` は **PerformanceProfile** YAML の **name** です (例: **performance-dynamic-irq-profile**)。
7. ノードセレクターを `cnf-worker` をターゲットに設定するように設定します。
8. Pod が正常に実行されていることを確認します。ステータスが **running** であり、正しい `cnf-worker` ノードが設定されている必要があります。

```
$ oc get pod -o wide
```

予想される出力

```
NAME          READY STATUS  RESTARTS  AGE   IP           NODE
NOMINATED NODE READINESS GATES
dynamic-irq-pod 1/1   Running  0         5h33m <ip-address> <node-name> <none>
```

```
<none>
```

9. IRQ の動的負荷分散向けに設定された Pod が実行される CPU を取得します。

```
$ oc exec -it dynamic-irq-pod -- /bin/bash -c "grep Cpus_allowed_list /proc/self/status | awk '{print $2}'"
```

予想される出力

```
Cpus_allowed_list: 2-3
```

10. ノードの設定が正しく適用されていることを確認します。ノードにログインして設定を確認します。

```
$ oc debug node/<node-name>
```

予想される出力

```
Starting pod/<node-name>-debug ...
To use host binaries, run `chroot /host`

Pod IP: <ip-address>
If you don't see a command prompt, try pressing enter.

sh-4.4#
```

11. ノードのファイルシステムを使用できることを確認します。

```
sh-4.4# chroot /host
```

予想される出力

```
sh-4.4#
```

12. デフォルトのシステム CPU アフィニティマスクに **dynamic-irq-pod** CPU(例: CPU 2 および 3) が含まれないようにします。

```
$ cat /proc/irq/default_smp_affinity
```

出力例

```
33
```

13. システム IRQ が **dynamic-irq-pod** CPU で実行されるように設定されていないことを確認します。

```
find /proc/irq/ -name smp_affinity_list -exec sh -c 'i="$1"; mask=$(cat $i); file=$(echo $i); echo $file: $mask' _ {} \;
```

出力例

```

/proc/irq/0/smp_affinity_list: 0-5
/proc/irq/1/smp_affinity_list: 5
/proc/irq/2/smp_affinity_list: 0-5
/proc/irq/3/smp_affinity_list: 0-5
/proc/irq/4/smp_affinity_list: 0
/proc/irq/5/smp_affinity_list: 0-5
/proc/irq/6/smp_affinity_list: 0-5
/proc/irq/7/smp_affinity_list: 0-5
/proc/irq/8/smp_affinity_list: 4
/proc/irq/9/smp_affinity_list: 4
/proc/irq/10/smp_affinity_list: 0-5
/proc/irq/11/smp_affinity_list: 0
/proc/irq/12/smp_affinity_list: 1
/proc/irq/13/smp_affinity_list: 0-5
/proc/irq/14/smp_affinity_list: 1
/proc/irq/15/smp_affinity_list: 0
/proc/irq/24/smp_affinity_list: 1
/proc/irq/25/smp_affinity_list: 1
/proc/irq/26/smp_affinity_list: 1
/proc/irq/27/smp_affinity_list: 5
/proc/irq/28/smp_affinity_list: 1
/proc/irq/29/smp_affinity_list: 0
/proc/irq/30/smp_affinity_list: 0-5

```

11.3.4. IRQ アフィニティ設定のサポートについて

一部の IRQ コントローラーでは IRQ アフィニティ設定がサポートされていないため、常にすべてのオンライン CPU が IRQ マスクとして公開されます。これらの IRQ コントローラーは CPU 0 で正常に実行されます。

以下は、IRQ アフィニティ設定がサポートされていないことを Red Hat が認識しているドライバーとハードウェアの例です。このリストはすべてを網羅しているわけではありません。

- **megaraid_sas** などの一部の RAID コントローラードライバー
- 多くの不揮発性メモリーエクスプレス (NVMe) ドライバー
- 一部の LAN on Motherboard (LOM) ネットワークコントローラー
- **managed_irqs** を使用するドライバー



注記

IRQ アフィニティ設定をサポートしない理由は、プロセッサの種類、IRQ コントローラー、マザーボードの回路接続などに関連している可能性があります。

分離された CPU に有効な IRQ アフィニティが設定されている場合は、一部のハードウェアまたはドライバーで IRQ アフィニティ設定がサポートされていないことを示唆している可能性があります。有効なアフィニティを見つけるには、ホストにログインし、次のコマンドを実行します。

```
$ find /proc/irq -name effective_affinity -printf "%p: " -exec cat {} \;
```

出力例

```
/proc/irq/0/effective_affinity: 1
/proc/irq/1/effective_affinity: 8
/proc/irq/2/effective_affinity: 0
/proc/irq/3/effective_affinity: 1
/proc/irq/4/effective_affinity: 2
/proc/irq/5/effective_affinity: 1
/proc/irq/6/effective_affinity: 1
/proc/irq/7/effective_affinity: 1
/proc/irq/8/effective_affinity: 1
/proc/irq/9/effective_affinity: 2
/proc/irq/10/effective_affinity: 1
/proc/irq/11/effective_affinity: 1
/proc/irq/12/effective_affinity: 4
/proc/irq/13/effective_affinity: 1
/proc/irq/14/effective_affinity: 1
/proc/irq/15/effective_affinity: 1
/proc/irq/24/effective_affinity: 2
/proc/irq/25/effective_affinity: 4
/proc/irq/26/effective_affinity: 2
/proc/irq/27/effective_affinity: 1
/proc/irq/28/effective_affinity: 8
/proc/irq/29/effective_affinity: 4
/proc/irq/30/effective_affinity: 4
/proc/irq/31/effective_affinity: 8
/proc/irq/32/effective_affinity: 8
/proc/irq/33/effective_affinity: 1
/proc/irq/34/effective_affinity: 2
```

一部のドライバーは、**managed_irqs** を使用します。そのアフィニティはカーネルによって内部的に管理され、ユーザー空間はアフィニティを変更できません。場合によっては、これらの IRQ が分離された CPU に割り当てられることもあります。**manage_irqs** の詳細については、[Affinity of managed interrupts cannot be changed even if they target isolated CPU](#) を参照してください。

11.3.5. クラスターのハイパースレッディングの設定

OpenShift Container Platform クラスターのハイパースレッディングを設定するには、パフォーマンスプロファイルの CPU スレッドを、予約または分離された CPU プールに設定された同じコアに設定します。



注記

パフォーマンスプロファイルを設定してから、ホストのハイパースレッディング設定を変更する場合は、新規の設定に一致するように **PerformanceProfile** YAML の CPU の **isolated** および **reserved** フィールドを更新するようにしてください。



警告

以前に有効にされたホストのハイパースレッディング設定を無効にすると、**PerformanceProfile** YAML にリスト表示されている CPU コア ID が正しくなくなる可能性があります。この設定が間違っていると、リスト表示される CPU が見つからなくなるため、ノードが利用できなくなる可能性があります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (oc) のインストール。

手順

1. 設定する必要があるホストのどの CPU でどのスレッドが実行されているかを確認します。クラスターにログインして以下のコマンドを実行し、ホスト CPU で実行されているスレッドを表示できます。

```
$ lscpu --all --extended
```

出力例

```
CPU NODE SOCKET CORE L1d:L1i:L2:L3 ONLINE MAXMHZ  MINMHZ
0 0 0 0 0:0:0:0 yes 4800.0000 400.0000
1 0 0 1 1:1:1:0 yes 4800.0000 400.0000
2 0 0 2 2:2:2:0 yes 4800.0000 400.0000
3 0 0 3 3:3:3:0 yes 4800.0000 400.0000
4 0 0 0 0:0:0:0 yes 4800.0000 400.0000
5 0 0 1 1:1:1:0 yes 4800.0000 400.0000
6 0 0 2 2:2:2:0 yes 4800.0000 400.0000
7 0 0 3 3:3:3:0 yes 4800.0000 400.0000
```

この例では、4つの物理 CPU コアで8つの論理 CPU コアが実行されています。CPU0 および CPU4 は物理コアの Core0 で実行されており、CPU1 および CPU5 は物理コア1で実行されています。

または、特定の物理 CPU コア (以下の例では **cpu0**) に設定されているスレッドを表示するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
$ cat /sys/devices/system/cpu/cpu0/topology/thread_siblings_list
```

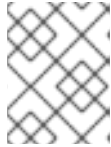
出力例

```
0-4
```

2. **PerformanceProfile** YAML で分離された CPU および予約された CPU を適用します。たとえば、論理コア CPU0 と CPU4 を **isolated** として設定し、論理コア CPU1 から CPU3 および CPU5 から CPU7 を **reserved** として設定できます。予約および分離された CPU を設定する場

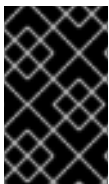
合に、Pod 内の infra コンテナは予約された CPU を使用し、アプリケーションコンテナは分離された CPU を使用します。

```
...
cpu:
  isolated: 0,4
  reserved: 1-3,5-7
...
```



注記

予約済みの CPU プールと分離された CPU プールは重複してはならず、これらは共に、ワーカーノードの利用可能なすべてのコアに広がる必要があります。



重要

ハイパースレッディングは、ほとんどの Intel プロセッサでデフォルトで有効にされます。ハイパースレッディングを有効にする場合、特定のコアによって処理されるスレッドはすべて、同じコアで分離されるか、処理される必要があります。

11.3.5.1. 低レイテンシーアプリケーションのハイパースレッディングの無効化

低レイテンシー処理用にクラスターを設定する場合、クラスターをデプロイする前にハイパースレッディングを無効にするかどうかを考慮してください。ハイパースレッディングを無効にするには、以下を実行します。

1. ハードウェアとトポロジーに適したパフォーマンスプロファイルを作成します。
2. **nosmt** を追加のカーネル引数として設定します。以下のパフォーマンスプロファイルの例は、この設定について示しています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - idle=poll
    - intel_idle.max_cstate=0
    - nosmt
  cpu:
    isolated: 2-3
    reserved: 0-1
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 2
      node: 0
      size: 1G
  nodeSelector:
```

```
node-role.kubernetes.io/performance: "
realTimeKernel:
  enabled: true
```

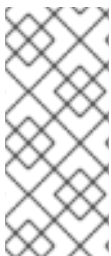


注記

予約および分離された CPU を設定する場合に、Pod 内の infra コンテナは予約された CPU を使用し、アプリケーションコンテナは分離された CPU を使用します。

11.3.6. ワークロードのヒントを理解する

次の表は、消費電力とリアルタイム設定の組み合わせがレイテンシーにどのように影響するかを示しています。



注記

次のワークロードヒントは手動で設定できます。Performance Profile Creator を使用して、ワークロードのヒントを操作することもできます。パフォーマンスプロファイルの詳細については、「パフォーマンスプロファイルの作成」セクションを参照してください。ワークロードヒントが手動で設定され、**realTime** ワークロードヒントが明示的に設定されていない場合は、デフォルトで **true** に設定されます。

パフォーマンスプロファイル作成者の設定	Hint	環境	説明
デフォルト	<pre>workloadHints: highPowerConsumption: false realTime: false</pre>	レイテンシー要件のない高スループットクラスター	CPU パーティショニングのみで達成されるパフォーマンス。
Low-latency	<pre>workloadHints: highPowerConsumption: false realTime: true</pre>	地域のデータセンター	エネルギー節約と低レイテンシーの両方が望ましい: 電力管理、レイテンシー、スループットの間での妥協。
Ultra-low-latency	<pre>workloadHints: highPowerConsumption: true realTime: true</pre>	ファーエッジクラスター、レイテンシークリティカルなワークロード	消費電力の増加を犠牲にして、絶対的な最小のレイテンシーと最大の決定論のために最適化されています。

パフォーマンスプロファイル作成者の設定	Hint	環境	説明
Pod ごとの電源管理	<pre>workloadHints: realTime: true highPowerConsumption: false perPodPowerManagement: true</pre>	重要なワークロードと重要でないワークロード	Pod ごとの電源管理が可能です。

関連情報

- Performance Profile Creator (PPC) を使用してパフォーマンスプロファイルを生成する方法の詳細は、[Creating a performance profile](#) を参照してください。

11.3.7. ワークロードヒントを手動で設定する

手順

- ワークロードのヒントについての表の説明に従って、環境のハードウェアとトポロジーに適した **PerformanceProfile** を作成します。予想されるワークロードに一致するようにプロファイルを調整します。この例では、可能な限り低いレイテンシーに調整します。
- highPowerConsumption** および **realTime** ワークロードのヒントを追加します。ここでは両方とも **true** に設定されています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: workload-hints
spec:
  ...
  workloadHints:
    highPowerConsumption: true ❶
    realTime: true ❷
```

- ❶ **highPowerConsumption** が **true** の場合、ノードは非常に低いレイテンシーに調整されますが、消費電力が増加します。
- ❷ システムの待ち時間に影響を与える可能性のある一部のデバッグおよび監視機能を無効にします。



注記

パフォーマンスプロファイルで **realTime** ワークロードヒントフラグが **true** に設定されている場合は、固定された CPU を持つすべての保証された Pod に **cpu-quota.crio.io: disable** アノテーションを追加します。このアノテーションは、Pod 内のプロセスのパフォーマンスの低下を防ぐために必要です。**realTime** ワークロードヒントが明示的に設定されていない場合は、デフォルトで **true** に設定されます。

関連情報

- 個々の保証された Pod の CPU スロットルを減らす方法は、[CPU CFS クォータの無効化](#) を参照してください。

11.3.8. infra およびアプリケーションコンテナの CPU の制限

一般的なハウスキーピングおよびワークロードタスクは、レイテンシーの影響を受けやすいプロセスに影響を与える可能性のある方法で CPU を使用します。デフォルトでは、コンテナランタイムはすべてのオンライン CPU を使用して、すべてのコンテナを一緒に実行します。これが原因で、コンテキストスイッチおよびレイテンシーが急増する可能性があります。CPU をパーティション化することで、ノイズの多いプロセスとレイテンシーの影響を受けやすいプロセスを分離し、干渉を防ぐことができます。以下の表は、Node Tuning Operator を使用してノードを調整した後、CPU でプロセスがどのように実行されるかを示しています。

表11.2 プロセスの CPU 割り当て

プロセスタイプ	Details
Burstable および BestEffort Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
インフラストラクチャー Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
割り込み	予約済み CPU にリダイレクトします (OpenShift Container Platform 4.7 以降ではオプション)
カーネルプロセス	予約済み CPU へのピン
レイテンシーの影響を受けやすいワークロード Pod	分離されたプールからの排他的 CPU の特定のセットへのピン
OS プロセス/systemd サービス	予約済み CPU へのピン

すべての QoS プロセスタイプ (**Burstable**、**BestEffort**、または **Guaranteed**) の Pod に割り当て可能なノード上のコアの容量は、分離されたプールの容量と同じです。予約済みプールの容量は、クラスターおよびオペレーティングシステムのハウスキーピング業務で使用するためにノードの合計コア容量から削除されます。

例 1

ノードは 100 コアの容量を備えています。クラスター管理者は、パフォーマンスプロファイルを使用して、50 コアを分離プールに割り当て、50 コアを予約プールに割り当てます。クラスター管理者は、25 コアを QoS **Guaranteed** Pod に割り当て、25 コアを **BestEffort** または **Burstable** Pod に割り当てます。これは、分離されたプールの容量と一致します。

例 2

ノードは 100 コアの容量を備えています。クラスター管理者は、パフォーマンスプロファイルを使用して、50 コアを分離プールに割り当て、50 コアを予約プールに割り当てます。クラスター管理者は、50 個のコアを QoS **Guaranteed** Pod に割り当て、1 個のコアを **BestEffort** または **Burstable** Pod に割り当てます。これは、分離されたプールの容量を 1 コア超えています。CPU 容量が不十分なため、Pod のスケジューリングが失敗します。

使用する正確なパーティショニングパターンは、ハードウェア、ワークロードの特性、予想されるシステム負荷などの多くの要因によって異なります。いくつかのサンプルユースケースは次のとおりです。

- レイテンシーの影響を受けやすいワークロードがネットワークインターフェイスコントローラー (NIC) などの特定のハードウェアを使用する場合は、分離されたプール内の CPU が、このハードウェアにできるだけ近いことを確認してください。少なくとも、ワークロードを同じ Non-Uniform Memory Access (NUMA) ノードに配置する必要があります。
- 予約済みプールは、すべての割り込みを処理するために使用されます。システムネットワークに依存する場合は、すべての着信パケット割り込みを処理するために、十分なサイズの予約プールを割り当てます。4.13 以降のバージョンでは、ワークロードはオプションで機密としてラベル付けできます。

予約済みパーティションと分離パーティションにどの特定の CPU を使用するかを決定するには、詳細な分析と測定が必要です。デバイスやメモリーの NUMA アフィニティーなどの要因が作用しています。選択は、ワークロードアーキテクチャーと特定のユースケースにも依存します。



重要

予約済みの CPU プールと分離された CPU プールは重複してはならず、これらは共に、ワーカーノードの利用可能なすべてのコアに広がる必要があります。

ハウスキーピングタスクとワークロードが相互に干渉しないようにするには、パフォーマンスプロファイルの **spec** セクションで CPU の 2 つのグループを指定します。

- **isolated** - アプリケーションコンテナワークロードの CPU を指定します。これらの CPU のレイテンシーが一番低くなります。このグループのプロセスには割り込みがないため、DPDK ゼロパケットロスの帯域幅がより高くなります。
- **reserved** - クラスタおよびオペレーティングシステムのハウスキーピング業務用の CPU を指定します。**reserved** グループのスレッドは、ビジーであることが多いです。**reserved** グループでレイテンシーの影響を受けやすいアプリケーションを実行しないでください。レイテンシーの影響を受けやすいアプリケーションは、**isolated** グループで実行されます。

手順

1. 環境のハードウェアとトポロジーに適したパフォーマンスプロファイルを作成します。
2. `infra` およびアプリケーションコンテナ用に予約して分離する CPU で、**reserved** および **isolated** パラメーターを追加します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: infra-cpus
spec:
  cpu:
    reserved: "0-4,9" ①
    isolated: "5-8" ②
  nodeSelector: ③
    node-role.kubernetes.io/worker: ""
```

- ① クラスタおよびオペレーティングシステムのハウスキーピングタスクを実行する `infra` コンテナの CPU を指定します。

- 2 アプリケーションコンテナがワークロードを実行する CPU を指定します。
- 3 オプション: ノードセレクターを指定してパフォーマンスプロファイルを特定のノードに適用します。

関連情報

- [Guaranteed Pod の分離された CPU のデバイス割り込み処理の管理](#)
- [Create a pod that gets assigned a QoS class of Guaranteed](#)

11.4. NODE TUNING OPERATOR を使用した NIC キューの削減

Node Tuning Operator を使用すると、各ネットワークデバイスのネットワークインターフェイスコントローラー (NIC) のキュー数を調整できます。PerformanceProfile を使用すると、キューの量を予約された CPU の数まで減らすことができます。

11.4.1. パフォーマンスプロファイルによる NIC キューの調整

パフォーマンスプロファイルを使用すると、各ネットワークデバイスのキュー数を調整できます。

サポート対象のネットワークデバイスは以下のとおりです。

- 非仮想ネットワークデバイス
- 複数のキュー (チャンネル) をサポートするネットワークデバイス

サポート対象外のネットワークデバイスは以下の通りです。

- Pure Software ネットワークインターフェイス
- ブロックデバイス
- Intel DPDK Virtual Function

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **cluster-admin** 権限を持つユーザーとして、Node Tuning Operator を実行する OpenShift Container Platform クラスターにログインします。
2. お使いのハードウェアとトポロジーに適したパフォーマンスプロファイルを作成して適用します。プロファイルの作成に関するガイダンスは、パフォーマンスプロファイルの作成のセクションを参照してください。
3. この作成したパフォーマンスプロファイルを編集します。

```
$ oc edit -f <your_profile_name>.yaml
```

4. **spec** フィールドに **net** オブジェクトを設定します。オブジェクトリストには、以下の2つのフィールドを含めることができます。
- **userLevelNetworking** は、ブール値フラグとして指定される必須フィールドです。**userLevelNetworking** が **true** の場合、サポートされているすべてのデバイスのキュー数は、予約された CPU 数に設定されます。デフォルトは **false** です。
 - **devices** は、キューを予約 CPU 数に設定するデバイスのリストを指定する任意のフィールドです。デバイスリストに何も指定しないと、設定がすべてのネットワークデバイスに適用されます。設定は以下のとおりです。
 - **InterfaceName**: このフィールドはインターフェイス名を指定し、正または負のシェルスタイルのワイルドカードをサポートします。
 - ワイルドカード構文の例: **<string>.***
 - 負のルールには、感嘆符のプリフィックスが付きます。除外リスト以外のすべてのデバイスにネットキューの変更を適用するには、**!<device>** を使用します (例: **!eno1**)。
 - **vendorID**: 16 ビット (16 進数) として表されるネットワークデバイスベンダー ID。接頭辞は **0x** です。
 - **deviceID**: 16 ビット (16 進数) として表されるネットワークデバイス ID (モデル)。接頭辞は **0x** です。



注記

deviceID が指定されている場合は、**vendorID** も定義する必要があります。デバイスエントリ **interfaceName**、**vendorID**、または **vendorID** と **deviceID** のペアで指定されているすべてのデバイス識別子に一致するデバイスは、ネットワークデバイスとしての資格があります。その後、このネットワークデバイスは net キュー数が予約 CPU 数に設定されます。

2つ以上のデバイスを指定すると、net キュー数は、それらのいずれかに一致する net デバイスに設定されます。

5. このパフォーマンスプロファイルの例を使用して、キュー数をすべてのデバイスの予約 CPU 数に設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

6. このパフォーマンスプロファイルの例を使用して、定義されたデバイス識別子に一致するすべてのデバイスの予約 CPU 数にキュー数を設定します。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  devices:
    - interfaceName: "eth0"
    - interfaceName: "eth1"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

7. このパフォーマンスプロファイルの例を使用して、インターフェイス名 **eth** で始まるすべてのデバイスの予約 CPU 数にキュー数を設定します。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  devices:
    - interfaceName: "eth*"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

8. このパフォーマンスプロファイルの例を使用して、**eno1** 以外の名前のインターフェイスを持つすべてのデバイスの予約 CPU 数にキュー数を設定します。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  devices:
    - interfaceName: "!eno1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

9. このパフォーマンスプロファイルの例を使用して、インターフェイス名 **eth0**、**0x1af4** の **vendorID**、および **0x1000** の **deviceID** を持つすべてのデバイスの予約 CPU 数にキュー数を設定します。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  devices:
    - interfaceName: "eth0"
      vendorID: "0x1af4"
      deviceID: "0x1000"
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""

```

10. 更新されたパフォーマンスプロファイルを適用します。

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

関連情報

- [パフォーマンスプロファイルの作成](#)。

11.4.2. キューステータスの確認

このセクションでは、さまざまなパフォーマンスプロファイルについて、変更の適用を検証する方法を複数例示しています。

例 1

この例では、サポートされている **すべての** デバイスの net キュー数は、予約された CPU 数 (2) に設定されます。

パフォーマンスプロファイルの関連セクションは次のとおりです。

```

apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
spec:
  cpu:
    reserved: 0-1 #total = 2
    isolated: 2-8
  net:
    userLevelNetworking: true
# ...

```

- 以下のコマンドを使用して、デバイスに関連付けられたキューのステータスを表示します。



注記

パフォーマンスプロファイルが適用されたノードで、以下のコマンドを実行します。

```
$ ethtool -l <device>
```

- プロファイルの適用前にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:      0
TX:      0
Other:   0
Combined: 4
Current hardware settings:
RX:      0
TX:      0
Other:   0
Combined: 4
```

- プロファイルの適用後にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:      0
TX:      0
Other:   0
Combined: 4
Current hardware settings:
RX:      0
TX:      0
Other:   0
Combined: 2 1
```

- 1** チャンネルを組み合わせると、すべてのサポート対象のデバイスの予約 CPU の合計数は 2 になります。これは、パフォーマンスプロファイルでの設定内容と一致します。

例 2

この例では、サポートされているすべてのネットワークデバイスの net キュー数は、予約された CPU 数 (2) に特定の **vendorID** を指定して、設定されます。

パフォーマンスプロファイルの関連セクションは次のとおりです。


```

apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1 #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
      devices:
        - vendorID = 0x1af4
# ...

```

- 以下のコマンドを使用して、デバイスに関連付けられたキューのステータスを表示します。



注記

パフォーマンスプロファイルが適用されたノードで、以下のコマンドを実行します。

```
$ ethtool -l <device>
```

- プロファイルの適用後にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```

Channel parameters for ens4:
Pre-set maximums:
RX:      0
TX:      0
Other:    0
Combined: 4
Current hardware settings:
RX:      0
TX:      0
Other:    0
Combined: 2 1

```

- 1** **vendorID=0x1af4** であるサポート対象の全デバイスの合計予約 CPU 数は 2 となります。たとえば、**vendorID=0x1af4** のネットワークデバイス **ens2** が別に存在する場合に、このデバイスも合計で 2 つの net キューを持ちます。これは、パフォーマンスプロファイルでの設定内容と一致します。

例 3

この例では、サポートされている **すべての** ネットワークデバイスが定義したデバイス ID のいずれかに一致する場合に、そのネットワークデバイスの net キュー数は、予約された CPU 数 (2) に設定されます。

udevadm info コマンドで、デバイスの詳細なレポートを確認できます。以下の例では、デバイスは以下ようになります。

```
# udevadm info -p /sys/class/net/ens4
```

```
...
E: ID_MODEL_ID=0x1000
E: ID_VENDOR_ID=0x1af4
E: INTERFACE=ens4
...
```

```
# udevadm info -p /sys/class/net/eth0
```

```
...
E: ID_MODEL_ID=0x1002
E: ID_VENDOR_ID=0x1001
E: INTERFACE=eth0
...
```

- **interfaceName** が **eth0** のデバイスの場合に net キューを 2 に、**vendorID=0x1af4** を持つデバイスには、以下のパフォーマンスプロファイルを設定します。

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1 #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
    devices:
      - interfaceName = eth0
      - vendorID = 0x1af4
  ...
```

- プロファイルの適用後にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:      0
TX:      0
Other:    0
Combined: 4
Current hardware settings:
RX:      0
TX:      0
Other:    0
Combined: 2 1
```

- 1 **vendorID=0x1af4** であるサポート対象の全デバイスの合計予約 CPU 数は 2 に設定されます。たとえば、**vendorID=0x1af4** のネットワークデバイス **ens2** が別に存在する場合に、このデバイスも合計で 2 つの net キューを持ちます。同様に、**interfaceName** が **eth0** のデバイスには、合計 net キューが 2 に設定されます。

11.4.3. NIC キューの調整に関するロギング

割り当てられたデバイスの詳細を示すログメッセージは、それぞれの Tuned デーモンログに記録されます。以下のメッセージは、`/var/log/tuned/tuned.log` ファイルに記録される場合があります。

- 正常に割り当てられたデバイスの詳細を示す **INFO** メッセージが記録されます。

```
INFO tuned.plugins.base: instance net_test (net): assigning devices ens1, ens2, ens3
```

- 割り当てることのできるデバイスがない場合は、**WARNING** メッセージが記録されます。

```
WARNING tuned.plugins.base: instance net_test: no matching devices available
```

11.5. 低レイテンシー CNF チューニングステータスのデバッグ

PerformanceProfile カスタムリソース (CR) には、チューニングのステータスを報告し、レイテンシーのパフォーマンスの低下の問題をデバッグするためのステータスフィールドが含まれます。これらのフィールドは、Operator の調整機能の状態を記述する状態について報告します。

パフォーマンスプロファイルに割り当てられるマシン設定プールのステータスが **degraded** 状態になると典型的な問題が発生する可能性があり、これにより **PerformanceProfile** のステータスが低下します。この場合、マシン設定プールは失敗メッセージを発行します。

Node Tuning Operator には **performanceProfile.spec.status.Conditions** ステータスフィールドが含まれています。

```
Status:
Conditions:
  Last Heartbeat Time: 2020-06-02T10:01:24Z
  Last Transition Time: 2020-06-02T10:01:24Z
  Status:              True
  Type:                 Available
  Last Heartbeat Time: 2020-06-02T10:01:24Z
  Last Transition Time: 2020-06-02T10:01:24Z
  Status:              True
  Type:                 Upgradeable
  Last Heartbeat Time: 2020-06-02T10:01:24Z
  Last Transition Time: 2020-06-02T10:01:24Z
  Status:              False
  Type:                 Progressing
  Last Heartbeat Time: 2020-06-02T10:01:24Z
  Last Transition Time: 2020-06-02T10:01:24Z
  Status:              False
  Type:                 Degraded
```

Status フィールドには、パフォーマンスプロファイルのステータスを示す **Type** 値を指定する **Conditions** が含まれます。

Available

すべてのマシン設定および Tuned プロファイルが正常に作成され、クラスターコンポーネントで利用可能になり、それら (NTO、MCO、Kubelet) を処理します。

Upgradeable

Operator によって維持されるリソースは、アップグレードを実行する際に安全な状態にあるかどうかを示します。

Progressing

パフォーマンスプロファイルからのデプロイメントプロセスが開始されたことを示します。

Degraded

以下の場合にエラーを示します。

- パフォーマンスプロファイルの検証に失敗しました。
- すべての関連するコンポーネントの作成が完了しませんでした。

これらのタイプには、それぞれ以下のフィールドが含まれます。

Status

特定のタイプの状態 (**true** または **false**)。

Timestamp

トランザクションのタイムスタンプ。

Reason string

マシンの読み取り可能な理由。

Message string

状態とエラーの詳細を説明する人が判読できる理由 (ある場合)。

11.5.1. マシン設定プール

パフォーマンスプロファイルとその作成される製品は、関連付けられたマシン設定プール (MCP) に従ってノードに適用されます。MCP は、カーネル引数、kube 設定、Huge Page の割り当て、および `rt-kernel` のデプロイメントを含むパフォーマンスプロファイルが作成するマシン設定の適用に関する進捗についての貴重な情報を保持します。パフォーマンスプロファイルコントローラーは MCP の変更を監視し、それに応じてパフォーマンスプロファイルのステータスを更新します。

MCP は、**Degraded** の場合に限りパフォーマンスプロファイルステータスに戻し、**performanceProfile.status.condition.Degraded = true** になります。

例

以下の例は、これに作成された関連付けられたマシン設定プール (**worker-cnf**) を持つパフォーマンスプロファイルのサンプルです。

1. 関連付けられたマシン設定プールの状態は `degraded` (低下) になります。

```
# oc get mcp
```

出力例

```
NAME          CONFIG          UPDATED  UPDATING  DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
```

```

DEGRADEDMACHINECOUNT AGE
master rendered-master-2ee57a93fa6c9181b546ca46e1571d2d True False
False 3 3 3 0 2d21h
worker rendered-worker-d6b2bdc07d9f5a59a6b68950acf25e5f True False
False 2 2 2 0 2d21h
worker-cnf rendered-worker-cnf-6c838641b8a08fff08dbd8b02fb63f7c False True
True 2 1 1 1 2d20h

```

2. MCP の **describe** セクションには理由が示されます。

```
# oc describe mcp worker-cnf
```

出力例

```

Message:      Node node-worker-cnf is reporting: "prepping update:
machineconfig.machineconfiguration.openshift.io \"rendered-worker-cnf-
40b9996919c08e335f3ff230ce1d170\" not
found"
Reason:      1 nodes are reporting degraded status on sync

```

3. degraded (低下) の状態は、**degraded = true** とマークされたパフォーマンスプロファイルの **status** フィールドにも表示されるはずです。

```
# oc describe performanceprofiles performance
```

出力例

```

Message: Machine config pool worker-cnf Degraded Reason: 1 nodes are reporting
degraded status on sync.
Machine config pool worker-cnf Degraded Message: Node yquinn-q8s5v-w-b-
z5lqn.c.openshift-gce-devel.internal is
reporting: "prepping update: machineconfig.machineconfiguration.openshift.io
\"rendered-worker-cnf-40b9996919c08e335f3ff230ce1d170\" not found". Reason:
MCPDegraded
Status: True
Type: Degraded

```

11.6. RED HAT サポート向けの低レイテンシーのチューニングデバッグデータの収集

サポートケースを作成する際、ご使用のクラスターについてのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

must-gather ツールを使用すると、ノードのチューニング、NUMA トポロジー、および低レイテンシーの設定に関する問題のデバッグに必要な OpenShift Container Platform クラスターについての診断情報を収集できます。

迅速なサポートを得るには、OpenShift Container Platform と低レイテンシーチューニングの両方の診断情報を提供してください。

11.6.1. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- 監査ログ
- サービスログ

--image 引数を指定してコマンドを実行する際にイメージを指定できます。イメージを指定する際、ツールはその機能または製品に関連するデータを収集します。**oc adm must-gather** を実行すると、新しい Pod がクラスターに作成されます。データは Pod で収集され、**must-gather.local** で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

11.6.2. 低レイテンシーチューニングデータの収集について

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、以下を始めとする低レイテンシーチューニングに関連する機能およびオブジェクトが含まれます。

- Node Tuning Operator namespace と子オブジェクト
- **MachineConfigPool** および関連付けられた **MachineConfig** オブジェクト
- Node Tuning Operator および関連付けられた Tuned オブジェクト
- Linux カーネルコマンドラインオプション
- CPU および NUMA トポロジー
- 基本的な PCI デバイス情報と NUMA 局所性

must-gather でデバッグ情報を収集するには、Performance Addon Operator **must-gather** イメージを指定する必要があります。

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.13.
```



注記

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、この機能は Node Tuning Operator の一部です。ただし、**must-gather** コマンドを実行するときは、引き続き **performance-addon-operator-must-gather** イメージを使用する必要があります。

11.6.3. 特定の機能に関するデータ収集

oc adm must-gather CLI コマンドを **--image** または **--image-stream** 引数と共に使用して、特定に機能についてのデバッグ情報を収集できます。**must-gather** ツールは複数のイメージをサポートするため、単一のコマンドを実行して複数の機能についてのデータを収集できます。



注記

特定の機能データに加えてデフォルトの **must-gather** データを収集するには、**--image-stream=openshift/must-gather** 引数を追加します。



注記

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 では、これらの機能は Node Tuning Operator の一部です。ただし、**must-gather** コマンドを実行するときは、引き続き **performance-addon-operator-must-gather** イメージを使用する必要があります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (oc) がインストールされている。

手順

1. **must-gather** データを保存するディレクトリーに移動します。
2. **oc adm must-gather** コマンドを1つまたは複数の **--image** または **--image-stream** 引数と共に実行します。たとえば、以下のコマンドは、デフォルトのクラスターデータと Node Tuning Operator に固有の情報の両方を収集します。

```
$ oc adm must-gather \
--image-stream=openshift/must-gather \ ❶
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.13 ❷
```

- ❶ デフォルトの OpenShift Container Platform **must-gather** イメージ。
 - ❷ 低レイテンシーチューニングの診断用の **must-gather** イメージ。
3. 作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ ❶
```

- ❶ **must-gather-local.5421342344627712289/** を実際のディレクトリー名に置き換えます。
4. 圧縮ファイルを [Red Hat カスタマーポータル](#) で作成したサポートケースに添付します。

関連情報

- MachineConfig および KubeletConfig についての詳細は、[ノードの管理](#) を参照してください。
- Node Tuning Operator の詳細は、[ノードチューニング Operator について](#) を参照してください。
- PerformanceProfile の詳細は、[Huge Page の設定](#) を参照してください。
- コンテナからの Huge Page の消費に関する詳細は、[How huge pages are consumed by apps](#) を参照してください。

第12章 プラットフォーム検証のためのレイテンシーテストの実行

Cloud-native Network Functions (CNF) テストイメージを使用して、CNF ワークロードの実行に必要なすべてのコンポーネントがインストールされている CNF 対応の OpenShift Container Platform クラスターでレイテンシーテストを実行できます。レイテンシーテストを実行して、ワークロードのノードチューニングを検証します。

cnf-tests コンテナイメージは、registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 で入手できます。



重要

cnf-tests イメージには、現時点で Red Hat がサポートしていないいくつかのテストも含まれています。Red Hat がサポートしているのはレイテンシーテストのみです。

12.1. レイテンシーテストを実行するための前提条件

レイテンシーテストを実行するには、クラスターが次の要件を満たしている必要があります。

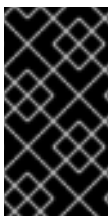
1. Node Tuning Operator を使用してパフォーマンスプロファイルを設定しました。
2. 必要なすべての CNF 設定をクラスターに適用しました。
3. クラスターに既存の **MachineConfigPool** CR が適用されている。デフォルトのワーカープールは **worker-cnf** です。

関連情報

- クラスターパフォーマンスプロファイルの作成の詳細は、[リアルタイム機能を使用したワーカーのプロビジョニング](#) を参照してください。

12.2. レイテンシーテストの検出モードについて

検出モードでは、設定を変更せずにクラスターの機能を検証できます。既存の環境設定はテストに使用されます。テストは、必要な設定アイテムを見つけ、それらのアイテムを使用してテストを実行できます。特定のテストの実行に必要なリソースが見つからない場合、テストは省略され、ユーザーに適切なメッセージが表示されます。テストが完了すると、事前に設定された設定項目のクリーンアップは行われず、テスト環境は別のテストの実行にすぐに使用できます。



重要

レイテンシーテストを実行するときは、必ず **-e DISCOVERY_MODE=true** および **-ginkgo.focus** を適切なレイテンシーテストに設定してテストを実行してください。遅延テストを検出モードで実行しない場合、既存のライブクラスターパフォーマンスプロファイル設定は、テストの実行によって変更されます。

テスト中に使用されるノードの制限

-e NODES_SELECTOR=node-role.kubernetes.io/worker-cnf などの **NODES_SELECTOR** 環境変数を指定することで、テストが実行されるノードを制限できます。テストによって作成されるリソースは、ラベルが一致するノードに限定されます。



注記

デフォルトのワーカープールをオーバーライドする場合は、適切なラベルを指定するコマンドに **-e ROLE_WORKER_CNF=<custom_worker_pool>** 変数を渡します。

12.3. レイテンシーの測定

cnf-tests イメージは、3つのツールを使用してシステムのレイテンシーを測定します。

- **hwlatdetect**
- **cyclictest**
- **oslat**

各ツールには特定の用途があります。信頼できるテスト結果を得るために、ツールを順番に使用します。

hwlatdetect

ベアメタルハードウェアが達成できるベースラインを測定します。次のレイテンシーテストに進む前に、**hwlatdetect** によって報告されるレイテンシーが必要なしきい値を満たしていることを確認してください。これは、オペレーティングシステムのチューニングによってハードウェアレイテンシーのスパイクを修正することはできないためです。

cyclictest

hwlatdetect が検証に合格した後、リアルタイムのカーネルスケジューラーのレイテンシーを検証します。**cyclictest** ツールは繰り返しタイマーをスケジュールし、希望のトリガー時間と実際のトリガーの時間の違いを測定します。この違いは、割り込みまたはプロセスの優先度によって生じるチューニングで、基本的な問題を発見できます。ツールはリアルタイムカーネルで実行する必要があります。

oslat

CPU 集約型 DPDK アプリケーションと同様に動作し、CPU の高いデータ処理をシミュレーションするビジーループにすべての中断と中断を測定します。

テストでは、次の環境変数が導入されます。

表12.1 レイテンシーテスト環境変数

環境変数	説明
LATENCY_TEST_DELAY	テストの実行を開始するまでの時間を秒単位で指定します。この変数を使用すると、CPU マネージャーの調整ループでデフォルトの CPU プールを更新できるようになります。デフォルト値は 0 です。
LATENCY_TEST_CPUS	レイテンシーテストを実行する Pod が使用する CPU の数を指定します。変数を設定しない場合、デフォルト設定にはすべての分離された CPU が含まれます。
LATENCY_TEST_RUNTIME	レイテンシーテストを実行する必要がある時間を秒単位で指定します。デフォルト値は 300 秒です。

環境変数	説明
HWLATDETECT_MAXIMUM_LATENCY	ワークロードとオペレーティングシステムの最大許容ハードウェアレイテンシーをマイクロ秒単位で指定します。 HWLATDETECT_MAXIMUM_LATENCY または MAXIMUM_LATENCY の値を設定しない場合、ツールはデフォルトの予想しきい値 (20µs) とツール自体の実際の最大レイテンシーを比較します。次に、テストはそれに応じて失敗または成功します。
CYCLICTEST_MAXIMUM_LATENCY	cyclictest の実行中に、ウェイクアップする前にすべてのスレッドが期待する最大レイテンシーをマイクロ秒単位で指定します。 CYCLICTEST_MAXIMUM_LATENCY または MAXIMUM_LATENCY の値を設定しない場合、ツールは予想される最大レイテンシーと実際の最大レイテンシーの比較をスキップします。
OSLAT_MAXIMUM_LATENCY	oslat テスト結果の最大許容レイテンシーをマイクロ秒単位で指定します。 OSLAT_MAXIMUM_LATENCY または MAXIMUM_LATENCY の値を設定しない場合、ツールは予想される最大レイテンシーと実際の最大レイテンシーの比較をスキップします。
MAXIMUM_LATENCY	最大許容レイテンシーをマイクロ秒単位で指定する統合変数。利用可能なすべてのレイテンシーツールに適用できます。
LATENCY_TEST_RUN	テストを実行するかどうかを示すブールパラメーター。 LATENCY_TEST_RUN はデフォルトで false に設定されています。レイテンシーテストを実行するには、この値を true に設定します。



注記

レイテンシーツールに固有の変数は、統合された変数よりも優先されます。たとえば、**OSLAT_MAXIMUM_LATENCY** が 30 マイクロ秒に設定され、**MAXIMUM_LATENCY** が 10 マイクロ秒に設定されている場合、**oslat** テストは 30 マイクロ秒の最大許容遅延で実行されます。

12.4. レイテンシーテストの実行

クラスターレイテンシーテストを実行して、クラウドネイティブネットワーク機能 (CNF) ワークロードのノードチューニングを検証します。



重要

遅延テストは常に **DISCOVERY_MODE=true** を設定して実行してください。そうしないと、テストスイートは実行中のクラスター設定に変更を加えます。



注記

非 root または非特権ユーザーとして **podman** コマンドを実行すると、パスのマウントが **permission denied** エラーで失敗する場合があります。**podman** コマンドを機能させるには、作成したボリュームに **:Z** を追加します。たとえば、**-v \$(pwd):/kubconfig:Z** です。これにより、**podman** は適切な SELinux の再ラベル付けを行うことができます。

手順

1. **kubeconfig** ファイルを含むディレクトリーでシェルプロンプトを開きます。
現在のディレクトリーにある **kubeconfig** ファイルとそれに関連する **\$KUBECONFIG** 環境変数を含むテストイメージを提供し、ボリュームを介してマウントします。これにより、実行中のコンテナがコンテナ内から **kubeconfig** ファイルを使用できるようになります。
2. 次のコマンドを入力して、レイテンシーテストを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

3. オプション: **-ginkgo.dryRun** を追加して、ドライランモードでレイテンシーテストを実行します。これは、テストの実行内容を確認するのに役立ちます。
4. オプション: **-ginkgo.v** を追加して、詳細度を上げてテストを実行します。
5. オプション: 特定のパフォーマンスプロファイルに対してレイテンシーテストを実行するには、次のコマンドを実行し、適切な値を置き換えます。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e FEATURES=performance -e
LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
-e PERF_TEST_PROFILE=<performance_profile> registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

ここでは、以下ようになります。

<performance_profile>

レイテンシーテストを実行するパフォーマンスプロファイルの名前です。

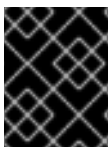


重要

有効なレイテンシーテストの結果を得るには、テストを少なくとも 12 時間実行します。

12.4.1. hwlatdetect の実行

hwlatdetect ツールは、Red Hat Enterprise Linux (RHEL) 9.x の通常のサブスクリプションを含む **rt-kernel** パッケージで利用できます。



重要

遅延テストは常に **DISCOVERY_MODE=true** を設定して実行してください。そうしないと、テストスイートは実行中のクラスター設定に変更を加えます。



注記

非 root または非特権ユーザーとして **podman** コマンドを実行すると、パスのマウントが **permission denied** エラーで失敗する場合があります。**podman** コマンドを機能させるには、作成したボリュームに **:Z** を追加します。たとえば、**-v \$(pwd)/:/kubecfg:Z** です。これにより、**podman** は適切な SELinux の再ラベル付けを行うことができます。

前提条件

- クラスタにリアルタイムカーネルをインストールしました。
- カスタマーポータル認証情報を使用して、**registry.redhat.io** にログインしました。

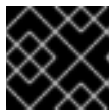
手順

- **hwlatdetect** テストを実行するには、変数値を適切に置き換えて、次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubecfg:Z -e KUBECONFIG=/kubecfg/kubecfg \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
e ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="hwlatdetect"
```

hwlatdetect テストは 10 分間 (600 秒) 実行されます。観測された最大レイテンシーが **MAXIMUM_LATENCY** (20 μ s) よりも低い場合、テストは正常に実行されます。

結果がレイテンシーのしきい値を超えると、テストは失敗します。



重要

有効な結果を得るには、テストを少なくとも 12 時間実行する必要があります。

障害出力の例

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0908 15:25:20.023712 27 request.go:601] Waited for 1.046586367s due to client-side
throttling, not priority and fairness, request:
GET:https://api.hlxcl6.lab.eng.tlv2.redhat.com:6443/apis/imageregistry.operator.openshift.io/v1?
timeout=32s
Running Suite: CNF Features e2e integration tests
=====
Random Seed: 1662650718
Will run 1 of 194 specs

[...]

• Failure [283.574 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
with the hwlatdetect image
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:228
```

```

should succeed [It]
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:236

Log file created at: 2022/09/08 15:25:27
Running on machine: hwlatdetect-b6n4n
Binary: Built with gc go1.17.12 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0908 15:25:27.160620    1 node.go:39] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd1,gpt3)/ostree/rhcos-
c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625769ef5fb9/vmlinuz-4.18.0-
372.19.1.el8_6.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ignition.platform.id=metal
ostree=/ostree/boot.1/rhcos/c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625
769ef5fb9/0 ip=dhcp root=UUID=5f80c283-f6e6-4a27-9b47-a287157483b2 rw
rootflags=prjquota boot=UUID=773bf59a-bafd-48fc-9a87-f62252d739d3 skew_tick=1
nohz=on rcu_nocbs=0-3 tuned.non_isolcpus=0000ffff,ffffff,ffffff0
systemd.cpu_affinity=4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79 intel_iommu=on iommu=pt
isolcpus=managed_irq,0-3 nohz_full=0-3 tsc=nowatchdog nosoftlockup nmi_watchdog=0
mce=off skew_tick=1 rcutree.kthread_prio=11 + +
I0908 15:25:27.160830    1 node.go:46] Environment information: kernel version 4.18.0-
372.19.1.el8_6.x86_64
I0908 15:25:27.160857    1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 100 --window
10000000us --width 950000us]
F0908 15:27:10.603523    1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect: test duration 100 seconds
  detector: tracer
  parameters:
    Latency threshold: 1us 1
    Sample window: 10000000us
    Sample width: 950000us
    Non-sampling period: 9050000us
    Output File: None

Starting test
test finished
Max Latency: 326us 2
Samples recorded: 5
Samples exceeding threshold: 5
ts: 1662650739.017274507, inner:6, outer:6
ts: 1662650749.257272414, inner:14, outer:326
ts: 1662650779.977272835, inner:314, outer:12
ts: 1662650800.457272384, inner:3, outer:9
ts: 1662650810.697273520, inner:3, outer:2

[...]

JUnit report was created: /junit.xml/cnftests-junit.xml

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the hwlatdetect image [It] should succeed

```

```
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:476
```

```
Ran 1 of 194 Specs in 365.797 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (366.08s)
FAIL
```

- 1 **MAXIMUM_LATENCY**または**HWLATDETECT_MAXIMUM_LATENCY**環境変数を使用して、レイテンシーしきい値を設定できます。
- 2 テスト中に測定される最大レイテンシー値。

hwlatdetect テスト結果の例

以下のタイプの結果をキャプチャーできます。

- テスト中に行われた変更への影響の履歴を作成するために、各実行後に収集される大まかな結果
- 最良の結果と設定を備えたラフテストの組み合わせセット

良い結果の例

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:
Latency threshold: 10us
Sample window: 1000000us
Sample width: 950000us
Non-sampling period: 50000us
Output File: None
```

```
Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
```

hwlatdetect ツールは、サンプルが指定されたしきい値を超えた場合にのみ出力を提供します。

悪い結果の例

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:Latency threshold: 10usSample window: 1000000us
Sample width: 950000usNon-sampling period: 50000usOutput File: None
```

```
Starting tests:1610542421.275784439, inner:78, outer:81
ts: 1610542444.330561619, inner:27, outer:28
ts: 1610542445.332549975, inner:39, outer:38
ts: 1610542541.568546097, inner:47, outer:32
ts: 1610542590.681548531, inner:13, outer:17
ts: 1610543033.818801482, inner:29, outer:30
ts: 1610543080.938801990, inner:90, outer:76
ts: 1610543129.065549639, inner:28, outer:39
```

```
ts: 1610543474.859552115, inner:28, outer:35
ts: 1610543523.973856571, inner:52, outer:49
ts: 1610543572.089799738, inner:27, outer:30
ts: 1610543573.091550771, inner:34, outer:28
ts: 1610543574.093555202, inner:116, outer:63
```

hwlatdetect の出力は、複数のサンプルがしきい値を超えていることを示しています。ただし、同じ出力は、次の要因に基づいて異なる結果を示す可能性があります。

- テストの期間
- CPU コアの数
- ホストファームウェアの設定



警告

次のレイテンシーテストに進む前に、**hwlatdetect** によって報告されたレイテンシーが必要なしきい値を満たしていることを確認してください。ハードウェアによって生じるレイテンシーを修正するには、システムベンダーのサポートに連絡しないといけない場合があります。

すべての遅延スパイクがハードウェアに関連しているわけではありません。ワークロードの要件を満たすようにホストファームウェアを調整してください。詳細は、[システムチューニング用のファームウェアパラメーターの設定](#) を参照してください。

12.4.2. cyclicttest の実行

cyclicttest ツールは、指定された CPU でのリアルタイムカーネルスケジューラーのレイテンシーを測定します。



重要

遅延テストは常に **DISCOVERY_MODE=true** を設定して実行してください。そうしないと、テストスイートは実行中のクラスター設定に変更を加えます。



注記

非 root または非特権ユーザーとして **podman** コマンドを実行すると、パスのマウントが **permission denied** エラーで失敗する場合があります。**podman** コマンドを機能させるには、作成したボリュームに **:Z** を追加します。たとえば、**-v \$(pwd)/:/kubconfig:Z** です。これにより、**podman** は適切な SELinux の再ラベル付けを行うことができます。

前提条件

- カスタマーポータル認証情報を使用して、**registry.redhat.io** にログインしました。
- クラスターにリアルタイムカーネルをインストールしました。

- Node Tuning Operator を使用してクラスターパフォーマンスプロファイルを適用しました。

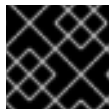
手順

- **cyclictest** を実行するには、次のコマンドを実行し、必要に応じて変数の値を置き換えます。

```
$ podman run -v $(pwd)/:/kubecfg:Z -e KUBECNF=/kubecfg/kubecfg \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
e ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="cyclictest"
```

このコマンドは、**cyclictest** ツールを 10 分 (600 秒) 実行します。観測された最大レイテンシーが **MAXIMUM_LATENCY** (この例では 20 μ s) よりも低い場合、テストは正常に実行されます。20 マイクロ秒以上の遅延スパイクは、一般に、通信事業者の RAN ワークロードでは受け入れられません。

結果がレイテンシーのしきい値を超えると、テストは失敗します。



重要

有効な結果を得るには、テストを少なくとも 12 時間実行する必要があります。

障害出力の例

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=cyclictest
I0908 13:01:59.193776 27 request.go:601] Waited for 1.046228824s due to client-side
throttling, not priority and fairness, request: GET:https://api.compute-
1.example.com:6443/apis/packages.operators.coreos.com/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=====
Random Seed: 1662642118
Will run 1 of 194 specs

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the cyclictest image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/func-tests/4_latency/latency.go:220

Ran 1 of 194 Specs in 161.151 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (161.48s)
FAIL
```

サイクルテスト結果の例

同じ出力は、ワークロードごとに異なる結果を示す可能性があります。たとえば、18 μ s までのスパイクは 4G DU ワークロードでは許容されますが、5G DU ワークロードでは許容されません。

良い結果の例


```

running cmd: cyclictst -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000001 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000002 579506 535967 418614 573648 532870 529897 489306 558076 582350 585188
583793 223781 532480 569130 472250 576043
More histogram entries ...
# Total: 000600000 000600000 000600000 000599999 000599999 000599999 000599998
000599998 000599998 000599997 000599997 000599996 000599996 000599995 000599995
000599995
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Max Latencies: 00005 00005 00004 00005 00004 00004 00005 00005 00006 00005 00004 00005
00004 00004 00005 00004
# Histogram Overflows: 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000 00000
# Histogram Overflow at cycle number:
# Thread 0:
# Thread 1:
# Thread 2:
# Thread 3:
# Thread 4:
# Thread 5:
# Thread 6:
# Thread 7:
# Thread 8:
# Thread 9:
# Thread 10:
# Thread 11:
# Thread 12:
# Thread 13:
# Thread 14:
# Thread 15:

```

悪い結果の例

```

running cmd: cyclictst -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
# Histogram
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000001 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000002 564632 579686 354911 563036 492543 521983 515884 378266 592621 463547
482764 591976 590409 588145 589556 353518
More histogram entries ...
# Total: 000599999 000599999 000599999 000599997 000599997 000599998 000599998
000599997 000599997 000599996 000599995 000599996 000599995 000599995 000599995
000599993
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002

```

```

00002 00002 00002 00002
# Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Max Latencies: 00493 00387 00271 00619 00541 00513 00009 00389 00252 00215 00539 00498
00363 00204 00068 00520
# Histogram Overflows: 00001 00001 00001 00002 00002 00001 00000 00001 00001 00001 00002
00001 00001 00001 00001 00002
# Histogram Overflow at cycle number:
# Thread 0: 155922
# Thread 1: 110064
# Thread 2: 110064
# Thread 3: 110063 155921
# Thread 4: 110063 155921
# Thread 5: 155920
# Thread 6:
# Thread 7: 110062
# Thread 8: 110062
# Thread 9: 155919
# Thread 10: 110061 155919
# Thread 11: 155918
# Thread 12: 155918
# Thread 13: 110060
# Thread 14: 110060
# Thread 15: 110059 155917

```

12.4.3. oslat の実行

oslat テストは、CPU を集中的に使用する DPDK アプリケーションをシミュレートし、すべての中断と中断を測定して、クラスターが CPU の負荷の高いデータ処理をどのように処理するかをテストします。



重要

遅延テストは常に **DISCOVERY_MODE=true** を設定して実行してください。そうしないと、テストスイートは実行中のクラスター設定に変更を加えます。



注記

非 root または非特権ユーザーとして **podman** コマンドを実行すると、パスのマウントが **permission denied** エラーで失敗する場合があります。**podman** コマンドを機能させるには、作成したボリュームに **:Z** を追加します。たとえば、**-v \$(pwd)/:/kubecfg:Z** です。これにより、**podman** は適切な SELinux の再ラベル付けを行うことができます。

前提条件

- カスタマーポータル認証情報を使用して、**registry.redhat.io** にログインしました。
- Node Tuning Operator を使用してクラスターパフォーマンスプロファイルを適用しました。

手順

- **oslat** テストを実行するには、変数値を適切に置き換えて、次のコマンドを実行します。

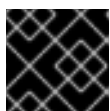
```
$ podman run -v $(pwd)/:/kubecfg:Z -e KUBECONFIG=/kubecfg/kubecfg \
```

```
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e FEATURES=performance -
e ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="oslat"
```

LATENCY_TEST_CPUS は、**oslat** コマンドでテストする CPU のリストを指定します。

このコマンドは、**oslat** ツールを 10 分 (600 秒) 実行します。観測された最大レイテンシーが **MAXIMUM_LATENCY** (20 μ s) よりも低い場合、テストは正常に実行されます。

結果がレイテンシーのしきい値を超えると、テストは失敗します。



重要

有効な結果を得るには、テストを少なくとも 12 時間実行する必要があります。

障害出力の例

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=oslat
I0908 12:51:55.999393 27 request.go:601] Waited for 1.044848101s due to client-side
throttling, not priority and fairness, request: GET:https://compute-
1.example.com:6443/apis/machineconfiguration.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
```

```
=====
Random Seed: 1662641514
Will run 1 of 194 specs
```

[...]

```
• Failure [77.833 seconds]
[performance] Latency Test
/remotesource/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
with the oslat image
/remotesource/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:128
should succeed [It]
/remotesource/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:153
```

The current latency 304 is bigger than the expected one 1 : **1**

[...]

Summarizing 1 Failure:

```
[Fail] [performance] Latency Test with the oslat image [It] should succeed
/remotesource/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:177
```

Ran 1 of 194 Specs in 161.091 seconds

```
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 193 Skipped
--- FAIL: TestTest (161.42s)
FAIL
```

- この例では、測定されたレイテンシーが最大許容値を超えています。

12.5. レイテンシーテストの失敗レポートの生成

次の手順を使用して、JUnit レイテンシーテストの出力とテストの失敗レポートを生成します。

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ログインしている。

手順

- レポートがダンプされる場所へのパスを **--report** パラメーターを渡すことで、クラスターの状態とトラブルシューティング用のリソースに関する情報を含むテスト失敗レポートを作成します。

```
$ podman run -v $(pwd)/:/kubecfg:Z -v $(pwd)/reportdest:<report_folder_path> \
-e KUBECFG=/kubecfg/kubecfg -e DISCOVERY_MODE=true -e
FEATURES=performance \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh --report <report_folder_path> \
-ginkgo.focus="[performance]\ Latency\ Test"
```

ここでは、以下のようになります。

<report_folder_path>

レポートが生成されるフォルダーへのパスです。

12.6. JUNIT レイテンシーテストレポートの生成

次の手順を使用して、JUnit レイテンシーテストの出力とテストの失敗レポートを生成します。

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ログインしている。

手順

- レポートがダンプされる場所へのパスとともに **--junit** パラメーターを渡すことにより、JUnit 準拠の XML レポートを作成します。

```
$ podman run -v $(pwd)/:/kubecfg:Z -v $(pwd)/junitdest:<junit_folder_path> \
-e KUBECFG=/kubecfg/kubecfg -e DISCOVERY_MODE=true -e
FEATURES=performance \
```

```
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh --junit <junit_folder_path> \
-ginkgo.focus="[performance]\ Latency\ Test"
```

ここでは、以下ようになります。

<junit_folder_path>

junit レポートが生成されるフォルダーへのパスです。

12.7. 単一ノードの OPENSIFT クラスタでレイテンシーテストを実行する

単一ノードの OpenShift クラスタでレイテンシーテストを実行できます。



重要

遅延テストは常に **DISCOVERY_MODE=true** を設定して実行してください。そうしないと、テストスイートは実行中のクラスタ設定に変更を加えます。



注記

非 root または非特権ユーザーとして **podman** コマンドを実行すると、パスのマウントが **permission denied** エラーで失敗する場合があります。**podman** コマンドを機能させるには、作成したボリュームに **:Z** を追加します。たとえば、**-v \$(pwd)/:/kubeconfig:Z** です。これにより、**podman** は適切な SELinux の再ラベル付けを行うことができます。

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ログインしている。

手順

- 単一ノードの OpenShift クラスタでレイテンシーテストを実行するには、次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e DISCOVERY_MODE=true -e FEATURES=performance -e
ROLE_WORKER_CNF=master \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```



注記

ROLE_WORKER_CNF=master は、ノードが所属する唯一のマシンプールであるため必須です。レイテンシーテストに必要な **MachineConfigPool** の設定は、レイテンシーテストを実行するための前提条件を参照してください。

テストスイートの実行後に、未解決のリソースすべてがクリーンアップされます。

12.8. 切断されたクラスタでのレイテンシーテストの実行

CNF テストイメージは、外部レジストリーに到達できない切断されたクラスターでテストを実行できます。これには、次の2つの手順が必要です。

1. **cnf-tests** イメージをカスタム切断レジストリーにミラーリングします。
2. カスタムの切断されたレジストリーからイメージを使用するようにテストに指示します。

クラスターからアクセスできるカスタムレジストリーへのイメージのミラーリング

mirror 実行ファイルがイメージに同梱されており、テストイメージをローカルレジストリーにミラーリングするために **oc** が必要とする入力を提供します。

1. クラスターおよび registry.redhat.io にアクセスできる中間マシンから次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
/usr/bin/mirror -registry <disconnected_registry> | oc image mirror -f -
```

ここでは、以下のようになります。

<disconnected_registry>

my.local.registry:5000/ など、設定した切断されたミラーレジストリーです。

2. **cnf-tests** イメージを切断されたレジストリーにミラーリングした場合は、テストの実行時にイメージの取得に使用された元のレジストリーをオーバーライドする必要があります。次に例を示します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e DISCOVERY_MODE=true -e FEATURES=performance -e IMAGE_REGISTRY="
<disconnected_registry>" \
-e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.13" \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

カスタムレジストリーからのイメージを使用するためのテストの設定

CNF_TESTS_IMAGE 変数と **IMAGE_REGISTRY** 変数を使用して、カスタムテストイメージとイメージレジストリーを使用してレイテンシーテストを実行できます。

- カスタムテストイメージとイメージレジストリーを使用するようにレイテンシーテストを設定するには、次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY="<custom_image_registry>" \
-e CNF_TESTS_IMAGE="<custom_cnf-tests_image>" \
-e FEATURES=performance \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 /usr/bin/test-run.sh
```

ここでは、以下のようになります。

<custom_image_registry>

custom.registry:5000/ などのカスタムイメージレジストリーです。

<custom_cnf-tests_image>

custom-cnf-tests-image:latest などのカスタム cnf-tests イメージです。

クラスター OpenShift イメージレジストリーへのイメージのミラーリング

OpenShift Container Platform は、クラスター上の標準ワークロードとして実行される組み込まれたコンテナイメージレジストリーを提供します。

手順

1. レジストリーをルートを使用して公開し、レジストリーへの外部アクセスを取得します。

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. 次のコマンドを実行して、レジストリーエンドポイントを取得します。

```
$ REGISTRY=$(oc get route default-route -n openshift-image-registry --template='{{.spec.host}}')
```

3. イメージを公開する namespace を作成します。

```
$ oc create ns cnftests
```

4. イメージストリームを、テストに使用されるすべての namespace で利用可能にします。これは、テスト namespace が **cnf-tests** イメージストリームからイメージを取得できるようにするために必要です。以下のコマンドを実行します。

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:cnf-features-testing:default --namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:performance-addon-operators-testing:default --namespace=cnftests
```

5. 次のコマンドを実行して、docker シークレット名と認証トークンを取得します。

```
$ SECRET=$(oc -n cnftests get secret | grep builder-docker | awk '{print $1}')
```

```
$ TOKEN=$(oc -n cnftests get secret $SECRET -o jsonpath="{.data[\".dockercfg\"]}" | base64 --decode | jq -r '.[\"image-registry.openshift-image-registry.svc:5000\"].auth')
```

6. **dockerauth.json** ファイルを作成します。次に例を示します。

```
$ echo '{"auths": { \"$REGISTRY\": { \"auth\": $TOKEN } } }' > dockerauth.json
```

7. イメージミラーリングを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:4.13 \
/usr/bin/mirror -registry $REGISTRY/cnftests | oc image mirror --insecure=true \
-a=$(pwd)/dockerauth.json -f -
```

8. テストを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e DISCOVERY_MODE=true -e FEATURES=performance -e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests \
```

```
cnf-tests-local:latest /usr/bin/test-run.sh -ginkgo.focus="\[performance\]\ Latency\ Test"
```

異なるテストイメージセットのミラーリング

オプションで、レイテンシーテスト用にミラーリングされるデフォルトのアップストリームイメージを変更できます。

手順

1. **mirror** コマンドは、デフォルトでアップストリームイメージをミラーリングしようとします。これは、以下の形式のファイルをイメージに渡すことで上書きできます。

```
[
  {
    "registry": "public.registry.io:5000",
    "image": "imageforcnfests:4.13"
  }
]
```

2. ファイルを **mirror** コマンドに渡します。たとえば、**images.json** としてローカルに保存します。以下のコマンドでは、ローカルパスはコンテナ内の **/kubecfg** にマウントされ、これを **mirror** コマンドに渡すことができます。

```
$ podman run -v $(pwd)/:/kubecfg:Z -e KUBECONFIG=/kubecfg/kubecfg \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubecfg/images.json" \
| oc image mirror -f -
```

12.9. CNF-TESTS コンテナでのエラーのトラブルシューティング

レイテンシーテストを実行するには、**cnf-tests** コンテナ内からクラスターにアクセス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ログインしている。

手順

- 次のコマンドを実行して、**cnf-tests** コンテナ内からクラスターにアクセスできることを確認します。

```
$ podman run -v $(pwd)/:/kubecfg:Z -e KUBECONFIG=/kubecfg/kubecfg \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.13 \
oc get nodes
```

このコマンドが機能しない場合は、DNS 間のスパン、MTU サイズ、またはファイアウォールアクセスに関連するエラーが発生している可能性があります。

第13章 ワーカーレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスタの安定性の向上

クラスタ管理者が遅延テストを実行してプラットフォームを検証した際に、遅延が大きい場合でも安定性を確保するために、クラスタの動作を調整する必要性が判明することがあります。クラスタ管理者が変更する必要があるのは、ファイルに記録されている1つのパラメーターだけです。このパラメーターは、監視プロセスがステータスを読み取り、クラスタの健全性を解釈する方法に影響を与える4つのパラメーターを制御するものです。1つのパラメーターのみを変更し、サポートしやすく簡単な方法でクラスタをチューニングできます。

Kubelet プロセスは、クラスタの健全性を監視する上での出発点です。**Kubelet** は、OpenShift Container Platform クラスタ内のすべてのノードのステータス値を設定します。Kubernetes コントローラマネージャー (**kube controller**) は、デフォルトで10秒ごとにステータス値を読み取ります。ノードのステータス値を読み取ることができない場合、設定期間が経過すると、**kube controller** とそのノードとの接続が失われます。デフォルトの動作は次のとおりです。

1. コントロールプレーン上のノードコントローラーが、ノードの健全性を **Unhealthy** に更新し、ノードの **Ready** 状態を `Unknown` とマークします。
2. この操作に応じて、スケジューラーはそのノードへの Pod のスケジューリングを停止します。
3. ノードライフサイクルコントローラーが、**NoExecute** effect を持つ **node.kubernetes.io/unreachable** ティントをノードに追加し、デフォルトでノード上のすべての Pod を5分後にエビクトするようにスケジュールします。

この動作は、ネットワークが遅延の問題を起こしやすい場合、特にネットワークエッジにノードがある場合に問題が発生する可能性があります。場合によっては、ネットワークの遅延が原因で、Kubernetes コントローラマネージャーが正常なノードから更新を受信できないことがあります。**Kubelet** は、ノードが正常であっても、ノードから Pod を削除します。

この問題を回避するには、**ワーカーレイテンシープロファイル** を使用して、**Kubelet** と Kubernetes コントローラマネージャーがアクションを実行する前にステータスの更新を待機する頻度を調整できます。これらの調整により、コントロールプレーンとワーカーノード間のネットワーク遅延が最適でない場合に、クラスタが適切に動作するようになります。

これらのワーカーレイテンシープロファイルには、3つのパラメーターセットが含まれています。パラメーターは、遅延の増加に対するクラスタの反応を制御するように、慎重に調整された値で事前定義されています。試験により手作業で最良の値を見つける必要はありません。

クラスタのインストール時、またはクラスタネットワークのレイテンシーの増加に気付いたときはいつでも、ワーカーレイテンシープロファイルを設定できます。

13.1. ワーカーレイテンシープロファイルについて

ワーカーレイテンシープロファイルは、4つの異なるカテゴリからなる慎重に調整されたパラメーターです。これらの値を実装する4つのパラメーターは、**node-status-update-frequency**、**node-monitor-grace-period**、**default-not-ready-toleration-seconds**、および **default-unreachable-toleration-seconds** です。これらのパラメーターにより、遅延の問題に対するクラスタの反応を制御できる値を使用できます。手作業で最適な値を決定する必要はありません。



重要

これらのパラメーターの手動設定はサポートされていません。パラメーター設定が正しくないと、クラスタの安定性に悪影響が及びます。

すべてのワーカーレイテンシープロファイルは、次のパラメーターを設定します。

node-status-update-frequency

kubelet がノードのステータスを API サーバーにポストする頻度を指定します。

node-monitor-grace-period

Kubernetes コントローラマネージャーが、ノードを異常とマークし、**node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントをノードに追加する前に、kubelet からの更新を待機する時間を秒単位で指定します。

default-not-ready-toleration-seconds

ノードを異常とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

default-unreachable-toleration-seconds

ノードを到達不能とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

次の Operator は、ワーカーレイテンシープロファイルの変更を監視し、それに応じて対応します。

- Machine Config Operator (MCO) は、ワーカーノードの **node-status-update-frequency** パラメーターを更新します。
- Kubernetes コントローラマネージャーは、コントロールプレーンノードの **node-monitor-grace-period** パラメーターを更新します。
- Kubernetes API Server Operator は、コントロールプレーンノードの **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** パラメーターを更新します。

ほとんどの場合はデフォルト設定が機能しますが、OpenShift Container Platform は、ネットワークで通常よりも高いレイテンシーが発生している状況に対して、他に 2 つのワーカーレイテンシープロファイルを提供します。次のセクションでは、3 つのワーカーレイテンシープロファイルについて説明します。

デフォルトのワーカーレイテンシープロファイル

Default プロファイルを使用すると、各 **Kubelet** が 10 秒ごとにステータスを更新します (**node-status-update-frequency**)。 **Kube Controller Manager** は、**Kubelet** のステータスを 5 秒ごとにチェックします (**node-monitor-grace-period**)。

Kubernetes コントローラマネージャーは、**Kubelet** が異常であると判断するまでに、**Kubelet** からのステータス更新を 40 秒待機します。ステータスが提供されない場合、Kubernetes コントローラマネージャーは、ノードに **node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントのマークを付け、そのノードの Pod を削除します。

そのノードの Pod に **NoExecute** テイントがある場合、その Pod は **tolerationSeconds** に従って実行されます。Pod にテイントがない場合、その Pod は 300 秒以内に削除されます (**Kube API Server** の **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** 設定)。

プロファイル	コンポーネント	パラメーター	値
デフォルト	kubelet	node-status-update-frequency	10s

プロファイル	コンポーネント	パラメーター	値
	Kubelet コントローラーマネージャー	node-monitor-grace-period	40s
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	300s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	300s

中規模のワーカーレイテンシープロファイル

ネットワークレイテンシーが通常の場合、**MediumUpdateAverageReaction** プロファイルを使用します。

MediumUpdateAverageReaction プロファイルは、kubelet の更新の頻度を 20 秒に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する期間を 2 分に変更します。そのノード上の Pod の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラーマネージャーは、ノードが異常であると判断するまでに 2 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
MediumUpdateAverageReaction	kubelet	node-status-update-frequency	20s
	Kubelet コントローラーマネージャー	node-monitor-grace-period	2m
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	60s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	60s

ワーカーの低レイテンシープロファイル

ネットワーク遅延が非常に高い場合は、**LowUpdateSlowReaction** プロファイルを使用します。

LowUpdateSlowReaction プロファイルは、kubelet の更新頻度を 1 分に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する時間を 5 分に変更します。そのノード上の Pod

の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラマネージャーは、ノードが異常であると判断するまでに 5 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
LowUpdateSlowReaction	kubelet	node-status-update-frequency	1m
	Kubelet コントローラマネージャー	node-monitor-grace-period	5m
	Kubernetes API Server Operator	default-not-ready-toleration-seconds	60s
	Kubernetes API Server Operator	default-unreachable-toleration-seconds	60s

13.2. クラスター作成時にワーカー遅延プロファイルを実装する



重要

インストーラーの設定を編集するには、まず **openshift-install create manifests** コマンドを使用して、デフォルトのノードマニフェストと他のマニフェスト YAML ファイルを作成する必要があります。このファイル構造を作成しなければ、`workerLatencyProfile` は追加できません。インストール先のプラットフォームにはさまざまな要件があります。該当するプラットフォームのドキュメントで、インストールセクションを参照してください。

workerLatencyProfile は、次の順序でマニフェストに追加する必要があります。

1. インストールに適したフォルダー名を使用して、クラスターの構築に必要なマニフェストを作成します。
2. **config.node** を定義する YAML ファイルを作成します。ファイルは **manifests** ディレクトリーに置く必要があります。
3. 初めてマニフェストで **workLatencyProfile** を定義する際には、クラスターの作成時に **Default**、**MediumUpdateAverageReaction**、または **LowUpdateSlowReaction** マニフェストのいずれかを指定します。

検証

- 以下は、マニフェストファイル内の **spec.workerLatencyProfile Default** 値を示すマニフェストを作成する例です。

```
$ openshift-install create manifests --dir=<cluster-install-dir>
```

- マニフェストを編集して値を追加します。この例では、**vi** を使用して、"Default" の **workerLatencyProfile** 値が追加されたマニフェストファイルの例を示します。

```
$ vi <cluster-install-dir>/manifests/config-node-default-profile.yaml
```

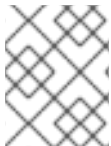
出力例

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  name: cluster
spec:
  workerLatencyProfile: "Default"
```

13.3. ワーカーレイテンシープロファイルの使用と変更

ネットワークの遅延に対処するためにワーカー遅延プロファイルを変更するには、**node.config** オブジェクトを編集してプロファイルの名前を追加します。遅延が増加または減少したときに、いつでもプロファイルを変更できます。

ワーカーレイテンシープロファイルは、一度に1つずつ移行する必要があります。たとえば、**Default** プロファイルから **LowUpdateSlowReaction** ワーカーレイテンシープロファイルに直接移行することはできません。まず **Default** ワーカーレイテンシープロファイルから **MediumUpdateAverageReaction** プロファイルに移行し、次に **LowUpdateSlowReaction** プロファイルに移行する必要があります。同様に、**Default** プロファイルに戻る場合は、まずロープロファイルからミディアムプロファイルに移行し、次に **Default** に移行する必要があります。



注記

OpenShift Container Platform クラスターのインストール時にワーカーレイテンシープロファイルを設定することもできます。

手順

デフォルトのワーカーレイテンシープロファイルから移動するには、以下を実行します。

1. 中規模のワーカーのレイテンシープロファイルに移動します。
 - a. **node.config** オブジェクトを編集します。

```
$ oc edit nodes.config/cluster
```

- b. **spec.workerLatencyProfile: MediumUpdateAverageReaction** を追加します。

node.config オブジェクトの例

```
apiVersion: config.openshift.io/v1
kind: Node
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: "2022-07-08T16:02:51Z"
generation: 1
name: cluster
ownerReferences:
- apiVersion: config.openshift.io/v1
  kind: ClusterVersion
  name: version
  uid: 36282574-bf9f-409e-a6cd-3032939293eb
resourceVersion: "1865"
uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction ❶

# ...

```

- ❶ 中規模のワーカーレイテンシーポリシーを指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

2. 必要に応じて、ワーカーのレイテンシーが低いプロファイルに移動します。
 - a. **node.config** オブジェクトを編集します。

```
$ oc edit nodes.config/cluster
```

- b. **spec.workerLatencyProfile** の値を **LowUpdateSlowReaction** に変更します。

node.config オブジェクトの例

```

apiVersion: config.openshift.io/v1
kind: Node
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: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: LowUpdateSlowReaction ❶

# ...

```

- 1 低ワーカーレイテンシーポリシーの使用を指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

検証

- 全ノードが **Ready** 状態に戻ると、以下のコマンドを使用して Kubernetes Controller Manager を確認し、これが適用されていることを確認できます。

```
$ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
```

出力例

```
# ...
- lastTransitionTime: "2022-07-11T19:47:10Z"
  reason: ProfileUpdated
  status: "False"
  type: WorkerLatencyProfileProgressing
- lastTransitionTime: "2022-07-11T19:47:10Z" 1
  message: all static pod revision(s) have updated latency profile
  reason: ProfileUpdated
  status: "True"
  type: WorkerLatencyProfileComplete
- lastTransitionTime: "2022-07-11T19:20:11Z"
  reason: AsExpected
  status: "False"
  type: WorkerLatencyProfileDegraded
- lastTransitionTime: "2022-07-11T19:20:36Z"
  status: "False"
# ...
```

- 1 プロファイルが適用され、アクティブであることを指定します。

ミディアムプロファイルからデフォルト、またはデフォルトからミディアムに変更する場合は、**node.config** オブジェクトを編集し、**spec.workerLatencyProfile** パラメーターを適切な値に設定します。

13.4. WORKERLATENCYPROFILE の結果の値を表示する手順の例

次のコマンドを使用して、**workerLatencyProfile** の値を表示できます。

検証

1. Kube API サーバーによる **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** フィールドの出力を確認します。

```
$ oc get KubeAPIServer -o yaml | grep -A 1 default-
```

出力例

```
default-not-ready-toleration-seconds:
```

```
- "300"  
default-unreachable-toleration-seconds:  
- "300"
```

2. Kube Controller Manager からの **node-monitor-grace-period** フィールドの値を確認します。

```
$ oc get KubeControllerManager -o yaml | grep -A 1 node-monitor
```

出力例

```
node-monitor-grace-period:  
- 40s
```

3. Kubelet からの **nodeStatusUpdateFrequency** 値を確認します。デバッグシェル内のルートディレクトリーとしてディレクトリー **/host** を設定します。root ディレクトリーを **/host** に変更すると、ホストの実行パスに含まれるバイナリーを実行できます。

```
$ oc debug node/<worker-node-name>  
$ chroot /host  
# cat /etc/kubernetes/kubelet.conf|grep nodeStatusUpdateFrequency
```

出力例

```
"nodeStatusUpdateFrequency": "10s"
```

これらの出力は、Worker Latency Profile のタイミング変数のセットを検証します。

第14章 パフォーマンスプロファイルの作成

Performance Profile Creator (PPC) ツールおよび、PPC を使用してパフォーマンスプロファイルを作成する方法を説明します。



注記

現在、CPU 負荷分散の無効化は cgroup v2 ではサポートされていません。その結果、cgroup v2 が有効になっている場合は、パフォーマンスプロファイルから望ましい動作が得られない可能性があります。パフォーマンスプロファイルを使用している場合は、cgroup v2 を有効にすることは推奨されません。

14.1. PERFORMANCE PROFILE CREATOR の概要

Performance Profile Creator (PPC) は、Node Tuning Operator に付属するコマンドラインツールで、パフォーマンスプロファイルを作成するために使用されます。このツールは、クラスターからの **must-gather** データと、ユーザー指定のプロファイル引数を複数使用します。PPC は、ハードウェアとトポロジーに適したパフォーマンスプロファイルを作成します。

このツールは、以下のいずれかの方法で実行します。

- **podman** の呼び出し
- ラッパースクリプトの呼び出し

14.1.1. must-gather コマンドを使用したクラスターに関するデータの収集

Performance Profile Creator (PPC) ツールには **must-gather** データが必要です。クラスター管理者は、**must-gather** コマンドを実行し、クラスターについての情報を取得します。



注記

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、この機能は Node Tuning Operator の一部です。ただし、**must-gather** コマンドを実行するときは、引き続き **performance-addon-operator-must-gather** イメージを使用する必要があります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- Performance Addon Operator へのアクセスは、イメージを **must gather** します。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. オプション: 一致するマシン設定プールがラベルを持つことを確認します。

```
$ oc describe mcp/worker-rt
```

出力例

-

```
Name:      worker-rt
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker-rt
```

- 一致するラベルが存在しない場合は、MCP 名と一致するマシン設定プール (MCP) のラベルを追加します。

```
$ oc label mcp <mcp_name> <mcp_name>=""
```

- must-gather** データを保存するディレクトリーに移動します。
- クラスターで **must-gather** を実行します。

```
$ oc adm must-gather --image=<PAO_must_gather_image> --dest-dir=<dir>
```



注記

must-gather コマンドは、**performance-addon-operator-must-gather** イメージを使用して実行する必要があります。この出力はオプションで圧縮できます。Performance Profile Creator ラッパースクリプトを実行している場合は、出力を圧縮する必要があります。

例

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.13 --dest-dir=<path_to_must-gather>/must-gather
```

- must-gather** ディレクトリーから圧縮ファイルを作成します。

```
$ tar cvaf must-gather.tar.gz must-gather/
```

14.1.2. podman を使用した Performance Profile Creator の実行

クラスター管理者は、**podman** および Performance Profile Creator を実行してパフォーマンスプロファイルを作成できます。

前提条件

- cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- ベアメタルハードウェアにインストールされたクラスター。
- podman** および OpenShift CLI (**oc**) がインストールされているノード。
- NodeTuningOperator イメージへのアクセス。

手順

- マシン設定プールを確認します。

```
$ oc get mcp
```

出力例

```

NAME          CONFIG          UPDATED  UPDATING  DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT AGE
master    rendered-master-acd1358917e9f98cbdb599aea622d78b    True    False
False    3          3          3          0          22h
worker-cnf rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826 False    True
False    2          1          1          0          22h

```

- Podman を使用して、**registry.redhat.io** への認証を行います。

```
$ podman login registry.redhat.io
```

```
Username: <username>
```

```
Password: <password>
```

- 必要に応じて、PPC ツールのヘルプを表示します。

```
$ podman run --rm --entrypoint performance-profile-creator registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13 -h
```

出力例

```
A tool that automates creation of Performance Profiles
```

```
Usage:
```

```
performance-profile-creator [flags]
```

```
Flags:
```

```

--disable-ht          Disable Hyperthreading
-h, --help           help for performance-profile-creator
--info string        Show cluster information; requires --must-gather-dir-path,
ignore the other arguments. [Valid values: log, json] (default "log")
--mcp-name string    MCP name corresponding to the target machines
(required)
--must-gather-dir-path string  Must gather directory path (default "must-gather")
--offlined-cpu-count int      Number of offlined CPUs
--per-pod-power-management    Enable Per Pod Power Management
--power-consumption-mode string  The power consumption mode. [Valid values:
default, low-latency, ultra-low-latency] (default "default")
--profile-name string      Name of the performance profile to be created (default
"performance")
--reserved-cpu-count int    Number of reserved CPUs (required)
--rt-kernel                Enable Real Time Kernel (required)
--split-reserved-cpus-across-numa  Split the Reserved CPUs across NUMA nodes
--topology-manager-policy string  Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
--user-level-networking    Run with User level Networking(DPDK) enabled

```

- Performance Profile Creator ツールを検出モードで実行します。



注記

検出モードは、**must-gather** からの出力を使用してクラスターを検査します。生成された出力には、以下のような情報が含まれます。

- 割り当てられた CPU ID でパーティションされた NUMA セル
- ハイパースレッディングが有効にされているかどうか

この情報を使用して、Performance Profile Creator ツールにわたす一部の引数に適切な値を設定できます。

```
$ podman run --entrypoint performance-profile-creator -v <path_to_must-gather>/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13 --info log --must-gather-dir-path /must-gather
```



注記

このコマンドは、Performance Profile Creator を、**podman** への新規エントリーポイントとして使用します。これは、ホストの **must-gather** データをコンテナイメージにマッピングし、ユーザーが提示した必須のプロファイル引数を呼び出し、**my-performance-profile.yaml** ファイルを生成します。

-v オプションでは、以下のいずれかへのパスを指定できます。

- **must-gather** 出力ディレクトリー
- **must-gather** のデプロイメント済みの tarball を含む既存のディレクトリー

info オプションでは、出力形式を指定する値が必要です。使用できる値は log と JSON です。JSON 形式はデバッグ用に確保されています。

5. podman を実行します。

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13 --mcp-name=worker-cnf --reserved-cpu-count=4 --rt-kernel=true --split-reserved-cpus-across- numa=false --must-gather-dir-path /must-gather --power-consumption-mode=ultra-low-latency --offlined-cpu-count=6 > my-performance-profile.yaml
```



注記

Performance Profile Creator の引数については Performance Profile Creator 引数の表に示しています。必要な引数は、以下の通りです。

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

この例の **mcp-name** 引数は、コマンド **oc get mcp** の出力に基づいて **worker-cnf** に設定されます。シングルノード OpenShift の場合は、**--mcp-name=master** を使用します。

6. 作成した YAML ファイルを確認します。

```
$ cat my-performance-profile.yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 2-39,48-79
    offlined: 42-47
    reserved: 0-1,40-41
  machineConfigPoolSelector:
    machineconfiguration.openshift.io/role: worker-cnf
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
  workloadHints:
    highPowerConsumption: true
    realTime: true
```

7. 生成されたプロファイルを適用します。

```
$ oc apply -f my-performance-profile.yaml
```

14.1.2.1. podman を実行してパフォーマンスプロファイルを作成する方法

以下の例では、**podman** を実行して、NUMA ノード間で分割される、予約済み CPU 20 個を指定してパフォーマンスプロファイルを作成する方法を説明します。

ノードのハードウェア設定:

- CPU 80 個
- ハイパースレッディングを有効にする
- NUMA ノード 2 つ
- NUMA ノード 0 に偶数個の CPU、NUMA ノード 1 に奇数個の CPU を稼働させる

podman を実行してパフォーマンスプロファイルを作成します。

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13 --mcp-name=worker-cnf --
reserved-cpu-count=20 --rt-kernel=true --split-reserved-cpus-across-numa=true --must-gather-dir-
path /must-gather > my-performance-profile.yaml
```

作成されたプロファイルは以下の YAML に記述されます。

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 10-39,50-79
    reserved: 0-9,40-49
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true

```



注記

この場合、CPU 10 個が NUMA ノード 0 に、残りの 10 個は NUMA ノード 1 に予約されます。

14.1.3. Performance Profile Creator ラッパースクリプトの実行

パフォーマンスプロファイルラッパースクリプトを使用すると、Performance Profile Creator (PPC) ツールの実行を簡素化できます。**podman** の実行に関連する煩雑性がなくなり、パフォーマンスプロファイルの作成が可能になります。

前提条件

- NodeTuningOperator イメージへのアクセス。
- **must-gather** tarball にアクセスできる。

手順

1. ローカルマシンにファイル (例: **run-perf-profile-creator.sh**) を作成します。

```
$ vi run-perf-profile-creator.sh
```

2. ファイルに以下のコードを貼り付けます。

```

#!/bin/bash

readonly CONTAINER_RUNTIME=${CONTAINER_RUNTIME:-podman}
readonly CURRENT_SCRIPT=$(basename "$0")
readonly CMD="${CONTAINER_RUNTIME} run --entrypoint performance-profile-creator"
readonly IMG_EXISTS_CMD="${CONTAINER_RUNTIME} image exists"
readonly IMG_PULL_CMD="${CONTAINER_RUNTIME} image pull"
readonly MUST_GATHER_VOL="/must-gather"

NTO_IMG="registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13"
MG_TARBALL=""
DATA_DIR=""

usage() {

```

```

print "Wrapper usage:"
print "  ${CURRENT_SCRIPT} [-h] [-p image][-t path] -- [performance-profile-creator flags]"
print ""
print "Options:"
print "  -h          help for ${CURRENT_SCRIPT}"
print "  -p          Node Tuning Operator image"
print "  -t          path to a must-gather tarball"

${IMG_EXISTS_CMD} "${NTO_IMG}" && ${CMD} "${NTO_IMG}" -h
}

function cleanup {
  [ -d "${DATA_DIR}" ] && rm -rf "${DATA_DIR}"
}
trap cleanup EXIT

exit_error() {
  print "error: $"
  usage
  exit 1
}

print() {
  echo "$*" >&2
}

check_requirements() {
  ${IMG_EXISTS_CMD} "${NTO_IMG}" || ${IMG_PULL_CMD} "${NTO_IMG}" || \
  exit_error "Node Tuning Operator image not found"

  [ -n "${MG_TARBALL}" ] || exit_error "Must-gather tarball file path is mandatory"
  [ -f "${MG_TARBALL}" ] || exit_error "Must-gather tarball file not found"

  DATA_DIR=$(mktemp -d -t "${CURRENT_SCRIPT}XXXX") || exit_error "Cannot create the
data directory"
  tar -zxf "${MG_TARBALL}" --directory "${DATA_DIR}" || exit_error "Cannot decompress the
must-gather tarball"
  chmod a+rx "${DATA_DIR}"

  return 0
}

main() {
  while getopts 'hp:t:' OPT; do
    case "${OPT}" in
      h)
        usage
        exit 0
        ;;
      p)
        NTO_IMG="${OPTARG}"
        ;;
      t)
        MG_TARBALL="${OPTARG}"
        ;;
      ?)

```

```

        exit_error "invalid argument: ${OPTARG}"
        ;;
    esac
done
shift $((OPTIND - 1))

check_requirements || exit 1

${CMD} -v "${DATA_DIR}:${MUST_GATHER_VOL}:z" "${NTO_IMG}" "$@" --must-gather-
dir-path "${MUST_GATHER_VOL}"
echo "" 1>&2
}

main "$@"

```

- このスクリプトの実行権限を全員に追加します。

```
$ chmod a+x run-perf-profile-creator.sh
```

- オプション: **run-perf-profile-creator.sh** コマンドの使用方法を表示します。

```
$/run-perf-profile-creator.sh -h
```

予想される出力

```

Wrapper usage:
run-perf-profile-creator.sh [-h] [-p image][-t path] -- [performance-profile-creator flags]

Options:
-h          help for run-perf-profile-creator.sh
-p          Node Tuning Operator image 1
-t          path to a must-gather tarball 2
A tool that automates creation of Performance Profiles

Usage:
performance-profile-creator [flags]

Flags:
--disable-ht          Disable Hyperthreading
-h, --help           help for performance-profile-creator
--info string        Show cluster information; requires --must-gather-dir-path,
ignore the other arguments. [Valid values: log, json] (default "log")
--mcp-name string    MCP name corresponding to the target machines
(required)
--must-gather-dir-path string  Must gather directory path (default "must-gather")
--offlined-cpu-count int      Number of offlined CPUs
--per-pod-power-management    Enable Per Pod Power Management
--power-consumption-mode string  The power consumption mode. [Valid values:
default, low-latency, ultra-low-latency] (default "default")
--profile-name string    Name of the performance profile to be created (default
"performance")
--reserved-cpu-count int      Number of reserved CPUs (required)
--rt-kernel                Enable Real Time Kernel (required)
--split-reserved-cpus-across-numa  Split the Reserved CPUs across NUMA nodes

```



```
--topology-manager-policy string  Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
--user-level-networking           Run with User level Networking(DPDK) enabled
```



注記

引数には、以下の2つのタイプがあります。

- ラッパー引数名は、**-h**、**-p**、および **-t** です。
- PPC 引数

- 1 オプション: Node Tuning Operator のイメージを指定します。設定されていない場合、デフォルトのアップストリームイメージが使用されます: **registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.13**。
- 2 **-t** は、必須のラッパースクリプトの引数で、**must-gather** tarball へのパスを指定します。

5. Performance Profile Creator ツールを検出モードで実行します。



注記

検出モードは、**must-gather** からの出力を使用してクラスターを検査します。生成された出力には、以下のような情報が含まれます。

- 割り当てられた CPU ID を使用した NUMA セルのパーティション設定
- ハイパースレッディングが有効にされているかどうか

この情報を使用して、Performance Profile Creator ツールにわたす一部の引数に適切な値を設定できます。

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --info=log
```



注記

info オプションでは、出力形式を指定する値が必要です。使用できる値は log と JSON です。JSON 形式はデバッグ用に確保されています。

6. マシン設定プールを確認します。

```
$ oc get mcp
```

出力例

```
NAME          CONFIG          UPDATED  UPDATING  DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT AGE
master       rendered-master-acd1358917e9f98cbdb599aea622d78b  True     False
```

False	3	3	3	0	22h		
worker-cnf	rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826	False	True				
False	2	1	1	0	22h		

7. パフォーマンスプロファイルを作成します。

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --mcp-name=worker-cnf --reserved-cpu-count=2 --rt-kernel=true > my-performance-profile.yaml
```



注記

Performance Profile Creator の引数については Performance Profile Creator 引数の表に示しています。必要な引数は、以下の通りです。

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

この例の **mcp-name** 引数は、コマンド **oc get mcp** の出力に基づいて **worker-cnf** に設定されます。シングルノード OpenShift の場合は、**--mcp-name=master** を使用します。

8. 作成した YAML ファイルを確認します。

```
$ cat my-performance-profile.yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 1-39,41-79
    reserved: 0,40
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: false
```

9. 生成されたプロファイルを適用します。





注記

プロファイルを適用する前に、Node Tuning Operator をインストールします。


```
$ oc apply -f my-performance-profile.yaml
```

14.1.4. Performance Profile Creator の引数

表14.1 Performance Profile Creator の引数

引数	説明
disable-ht	<p>ハイパースレッディングを無効にします。</p> <p>使用できる値は true または false です。</p> <p>デフォルト: false。</p> <div style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <p> 警告</p> <p>この引数が true に設定されている場合は、BIOS でハイパースレッディングを無効にしないでください。ハイパースレッディングの無効化は、カーネルコマンドライン引数で実行できます。</p> </div>
info	<p>この引数では、クラスター情報を取得します。使用できるのは検出モードのみです。検出モードでは、must-gather-dir-path 引数も必要です。他の引数が設定されている場合は無視されます。</p> <p>以下の値を使用できます。</p> <ul style="list-style-type: none"> ● log ● JSON <div style="display: flex; align-items: center;">  <p>注記</p> </div> <p>これらのオプションでは、デバッグ用に予約される JSON 形式で出力形式を定義します。</p> <p>デフォルト: log。</p>
mcp-name	<p>ターゲットマシンに対応する worker-cnf などの MCP 名。このパラメーターは必須です。</p>
must-gather-dir-path	<p>must gather のディレクトリーパス。このパラメーターは必須です。</p> <p>ラッパースクリプトでツールを実行する場合には、must-gather はスクリプト自体で指定されるので、ユーザーは指定しないでください。</p>

引数	説明
offlined-cpu-count	<p>オフラインの CPU の数。</p>  <p>注記</p> <p>これは 0 より大きい自然数でなければなりません。十分な数の論理プロセッサがオフラインにされていない場合、エラーメッセージがログに記録されます。メッセージは次のとおりです。</p> <p>Error: failed to compute the reserved and isolated CPUs: please ensure that reserved-cpu-count plus offlined-cpu-count should be in the range [0,1]</p> <p>Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]</p>
power-consumption-mode	<p>電力消費モード。</p> <p>以下の値を使用できます。</p> <ul style="list-style-type: none"> ● default: 有効な電力管理と基本的な低遅延を備えた CPU パーティション。 ● low-latency: レイテンシーの数値を改善するための強化された対策。 ● ultra-low-latency: 電力管理を犠牲にして、最適な遅延を優先します。 <p>デフォルト: default。</p>
per-pod-power-management	<p>Pod ごとの電源管理を有効にします。電力消費モードとして Ultra-low-latency を設定している場合、この引数は使用できません。</p> <p>使用できる値は true または false です。</p> <p>デフォルト: false。</p>
profile-name	<p>作成するパフォーマンスプロファイルの名前。デフォルト: performance。</p>

引数	説明
reserved-cpu-count	<p>予約された CPU の数。このパラメーターは必須です。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>注記</p> <p>これは自然数でなければなりません。0 の値は使用できません。</p> </div> </div>
rt-kernel	<p>リアルタイムカーネルを有効にします。このパラメーターは必須です。</p> <p>使用できる値は true または false です。</p>
split-reserved-cpus-across- numa	<p>NUMA ノード全体で予約された CPU を分割します。</p> <p>使用できる値は true または false です。</p> <p>デフォルト: false。</p>
topology-manager-policy	<p>作成するパフォーマンスプロファイルの kubelet Topology Manager ポリシー。</p> <p>以下の値を使用できます。</p> <ul style="list-style-type: none"> ● single-numa-node ● best-effort ● restricted <p>デフォルト: restricted。</p>
user-level-networking	<p>ユーザーレベルのネットワーク (DPDK) を有効にして実行します。</p> <p>使用できる値は true または false です。</p> <p>デフォルト: false。</p>

14.2. パフォーマンスプロファイルの参照

14.2.1. OpenStack で OVS-DPDK を使用するクラスター用のパフォーマンスプロファイルテンプレート

Red Hat OpenStack Platform (RHOSP) で Open vSwitch と Data Plane Development Kit (OVS-DPDK) を使用するクラスターでマシンのパフォーマンスを最大化するには、パフォーマンスプロファイルを使用できます。

次のパフォーマンスプロファイルテンプレートを使用して、デプロイメント用のプロファイルを作成できます。

OVS-DPDK を使用するクラスターのパフォーマンスプロファイルテンプレート

apiVersion: performance.openshift.io/v2

```
kind: PerformanceProfile
metadata:
  name: cnf-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - idle=poll
    - intel_idle.max_cstate=0
    - default_hugepagesz=1GB
    - hugepagesz=1G
    - intel_iommu=on
  cpu:
    isolated: <CPU_ISOLATED>
    reserved: <CPU_RESERVED>
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: <HUGEPAGES_COUNT>
      node: 0
      size: 1G
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  realTimeKernel:
    enabled: false
  globallyDisableIrqLoadBalancing: true
```

CPU_ISOLATED キー、**CPU_RESERVED** キー、および **HUGEPAGES_COUNT** キーの設定に適した値を入力します。

パフォーマンスプロファイルを作成および使用する方法については、OpenShift Container Platform ドキュメントのスケラビリティとパフォーマンスセクションのパフォーマンスプロファイルの作成ページを参照してください。

14.3. 関連情報

- **must-gather** ツールの詳細は、[Gathering data about your cluster](#) を参照してください。

第15章 ワークロードの分割

重要

ワークロードの分割はテクノロジープレビュー機能としてのみ使用できます。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。

リソースに制約のある環境では、ワークロードの分割を使用して、OpenShift Container Platform サービス、クラスター管理ワークロード、インフラストラクチャー Pod を分離し、予約済みの CPU セットで実行できます。

クラスター管理に必要な予約済み CPU の最小数は、4 つの CPU ハイパースレッド (HT) です。ワークロード分割では、クラスター管理ワークロードパーティションに含めるために、一連のクラスター管理 Pod と一連の一般的なアドオン Operator に注釈を付けます。これらの Pod は、最低限のサイズの CPU 設定内で正常に動作します。最小クラスター管理 Pod のセット外の追加の Operator またはワークロードでは、追加の CPU をワークロードパーティションに追加する必要があります。

ワークロード分割は、標準の Kubernetes スケジューリング機能を使用して、ユーザーワークロードをプラットフォームワークロードから分離します。

ワークロードの分割には次の変更が必要です。

1. `install-config.yaml` ファイルに、`cpuPartitioningMode` を追加フィールドとして追加します。

```
apiVersion: v1
baseDomain: devcluster.openshift.com
cpuPartitioningMode: AllNodes ❶
compute:
  - architecture: amd64
    hyperthreading: Enabled
    name: worker
    platform: {}
    replicas: 3
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform: {}
  replicas: 3
```

- ❶ インストール時に CPU のパーティション設定用クラスターをセットアップします。デフォルト値は **None** です。



注記

ワークロードの分割は、クラスターのインストール中にのみ有効にできます。インストール後にワークロードパーティショニングを無効にすることはできません。

2. パフォーマンスプロファイルで、**isolated** および **reserved** CPU を指定します。

推奨されるパフォーマンスプロファイル設定

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "module_blacklist=irdma"
  cpu:
    isolated: 2-51,54-103
    reserved: 0-1,52-53
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 32
      size: 1G
      node: 0
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
  workloadHints:
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: false

```

表15.1 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="868 1059 971 1346" style="background-color: black; width: 65px; height: 128px; margin-bottom: 10px;"></div> <p>重要</p> <p>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。

ワークロードパーティショニングにより、プラットフォーム Pod に拡張された **Management.workload.openshift.io/cores** リソースタイプが導入されます。kubelet は、対応するリソース内のプールに割り当てられた Pod でリソースと CPU リクエストをアダプタイズします。ワークロードの分割が有効になっている場合、スケジューラーは **management.workload.openshift.io/cores** リソースにより、デフォルトの **cpuset** だけでなく、ホストの **cpushares** 容量に基づいて Pod を適切に割り当てることができます。

関連情報

- 単一ノードの OpenShift クラスターで推奨される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。

第16章 NODE OBSERVABILITY OPERATOR を使用した CRI-O および KUBELET プロファイリングデータのリクエスト

Node Observability Operator は、ワーカーノードの CRI-O および Kubelet プロファイリングデータを収集して保存します。プロファイリングデータをクエリーして、CRI-O と Kubelet のパフォーマンスの傾向を分析し、パフォーマンス関連の問題をデバッグできます。



重要

Node Observability Operator 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。

16.1. NODE OBSERVABILITY OPERATOR のワークフロー

次のワークフローは、Node Observability Operator を使用してプロファイリングデータをクエリーする方法の概要を示しています。

1. Node Observability Operator を OpenShift Container Platform クラスタにインストールします。
2. NodeObservability カスタムリソースを作成して、選択したワーカーノードで CRI-O プロファイリングを有効にします。
3. プロファイリングクエリーを実行して、プロファイリングデータを生成します。

16.2. NODE OBSERVABILITY OPERATOR のインストール

Node Observability Operator は、デフォルトでは OpenShift Container Platform にインストールされていません。OpenShift Container Platform CLI または Web コンソールを使用して、Node Observability Operator をインストールできます。

16.2.1. CLI を使用した Node Observability Operator のインストール

OpenShift CLI(oc) を使用して、Node Observability Operator をインストールできます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- **cluster-admin** 権限でクラスタにアクセスできる。

手順

1. 次のコマンドを実行して、Node Observability Operator が使用可能であることを確認します。

```
$ oc get packagemanifests -n openshift-marketplace node-observability-operator
```

出力例

```
NAME                CATALOG          AGE
node-observability-operator  Red Hat Operators  9h
```

- 次のコマンドを実行して、**node-observability-operator** namespace を作成します。

```
$ oc new-project node-observability-operator
```

- OperatorGroup** オブジェクト YAML ファイルを作成します。

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  targetNamespaces: []
EOF
```

- Subscription** オブジェクトの YAML ファイルを作成して、namespace を Operator にサブスクライブします。

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  channel: alpha
  name: node-observability-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

検証

- 次のコマンドを実行して、インストールプラン名を表示します。

```
$ oc -n node-observability-operator get sub node-observability-operator -o yaml | yq
'.status.installplan.name'
```

出力例

```
install-dt54w
```

- 次のコマンドを実行して、インストールプランのステータスを確認します。

```
$ oc -n node-observability-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

<install_plan_name> は、前のコマンドの出力から取得したインストール計画名です。

出力例

```
COMPLETE
```

3. Node Observability Operator が稼働していることを確認します。

```
$ oc get deploy -n node-observability-operator
```

出力例

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
node-observability-operator-controller-manager	1/1	1	1	40h

16.2.2. Web コンソールを使用した Node Observability Operator のインストール

Node Observability Operator は、OpenShift Container Platform コンソールからインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. 管理者のナビゲーションパネルで、**Operators** → **OperatorHub** をデプロイメントします。
3. **All items** フィールドに **Node Observability Operator** と入力し、**Node Observability Operator** タイルを選択します。
4. **Install** をクリックします。
5. **Install Operator** ページで、次の設定を設定します。
 - a. **Update channel** 領域で、**alpha** をクリックします。
 - b. **Installation mode** 領域で、**A specific namespace on the cluster** をクリックします。
 - c. **Installed Namespace** リストから、リストから **node-observability-operator** を選択します。
 - d. **Update approval** 領域で、**Automatic** を選択します。
 - e. **Install** をクリックします。

検証

1. 管理者のナビゲーションパネルで、**Operators** → **Installed Operators** をデプロイメントします。
2. Node Observability Operator が Operators リストにリストされていることを確認します。

16.3. NODE OBSERVABILITY カスタムリソースの作成

プロファイリングクエリーを実行する前に、**NodeObservability** カスタムリソース (CR) を作成して実行する必要があります。一致する**NodeObservability** CR を実行すると、必要なマシン設定およびマシン設定プール CR が作成され、**nodeSelector** に一致するワーカーノードで CRI-O プロファイリングを有効にします。



重要

ワーカーノードで CRI-O プロファイリングが有効になっていない場合、**NodeObservabilityMachineConfig** リソースが作成されます。**NodeObservability** CR で指定された **nodeSelector** に一致するワーカーノードが再起動します。完了するまでに 10 分以上かかる場合があります。



注記

Kubelet プロファイリングはデフォルトで有効になっています。

ノードの CRI-Ounix ソケットは、エージェント Pod にマウントされます。これにより、エージェントは CRI-O と通信して pprof 要求を実行できます。同様に、**kubelet-serving-ca** 証明書チェーンはエージェント Pod にマウントされ、エージェントとノードの kubelet エンドポイント間の安全な通信を可能にします。

前提条件

- Node Observability Operator をインストールしました。
- OpenShift CLI (oc) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、OpenShift Container Platform CLI にログインします。

```
$ oc login -u kubeadmin https://<HOSTNAME>:6443
```

2. 次のコマンドを実行して、**node-observability-operator** namespace に切り替えます。

```
$ oc project node-observability-operator
```

3. 次のテキストを含む **nodeobservability.yaml** という名前の CR ファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
  name: cluster 1
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_hostname> 2
  type: crio-kubelet
```

- 1 クラスターごとに **NodeObservability** CR が1つしかないため、名前を **cluster** として指定する必要があります。
- 2 Node Observability エージェントをデプロイする必要があるノードを指定します。

4. **NodeObservability** CR を実行します。

```
oc apply -f nodeobservability.yaml
```

出力例

```
nodeobservability.olm.openshift.io/cluster created
```

5. 次のコマンドを実行して、**NodeObservability** CR のステータスを確認します。

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

出力例

```
conditions:
conditions:
- lastTransitionTime: "2022-07-05T07:33:54Z"
  message: 'DaemonSet node-observability-ds ready: true NodeObservabilityMachineConfig
  ready: true'
  reason: Ready
  status: "True"
  type: Ready
```

NodeObservability CR の実行は、理由が **Ready** で、ステータスが **True** のときに完了します。

16.4. プロファイリングクエリーの実行

プロファイリングクエリーを実行するには、**NodeObservabilityRun** リソースを作成する必要があります。プロファイリングクエリーは、CRI-O および Kubelet プロファイリングデータを 30 秒間フェッチするブロッキング操作です。プロファイリングクエリーが完了したら、コンテナファイルシステムの **/run/node-observability** ディレクトリー内のプロファイリングデータを取得する必要があります。データの有効期間は、**emptyDir** ボリュームを介してエージェント Pod にバインドされるため、エージェント Pod が **running** の状態にある間にプロファイリングデータにアクセスできます。



重要

一度にリクエストできるプロファイリングクエリーは1つだけです。

前提条件

- Node Observability Operator をインストールしました。
- **NodeObservability** カスタムリソース (CR) を作成しました。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 次のテキストを含む **nodeobservabilityrun.yaml** という名前の **NodeObservabilityRun** リソースファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun
spec:
  nodeObservabilityRef:
    name: cluster
```

2. **NodeObservabilityRun** リソースを実行して、プロファイリングクエリーをトリガーします。

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

3. 次のコマンドを実行して、**NodeObservabilityRun** のステータスを確認します。

```
$ oc get nodeobservabilityrun nodeobservabilityrun -o yaml | yq '.status.conditions'
```

出力例

```
conditions:
- lastTransitionTime: "2022-07-07T14:57:34Z"
  message: Ready to start profiling
  reason: Ready
  status: "True"
  type: Ready
- lastTransitionTime: "2022-07-07T14:58:10Z"
  message: Profiling query done
  reason: Finished
  status: "True"
  type: Finished
```

ステータスが **True** になり、タイプが **Finished** になると、プロファイリングクエリーは完了です。

4. 次の **bash** スクリプトを実行して、コンテナの **/run/node-observability** パスからプロファイリングデータを取得します。

```
for a in $(oc get nodeobservabilityrun nodeobservabilityrun -o yaml | yq
.status.agents[].name); do
  echo "agent ${a}"
  mkdir -p "/tmp/${a}"
  for p in $(oc exec "${a}" -c node-observability-agent -- bash -c "ls /run/node-
observability/*.pprof"); do
    f="$(basename ${p})"
    echo "copying ${f} to /tmp/${a}/${f}"
    oc exec "${a}" -c node-observability-agent -- cat "${p}" > "/tmp/${a}/${f}"
  done
done
```


第17章 ネットワーク遠端のクラスター

17.1. ネットワークファー遠端の課題

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

17.1.1. ネットワークファーエッジの課題を克服する

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

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

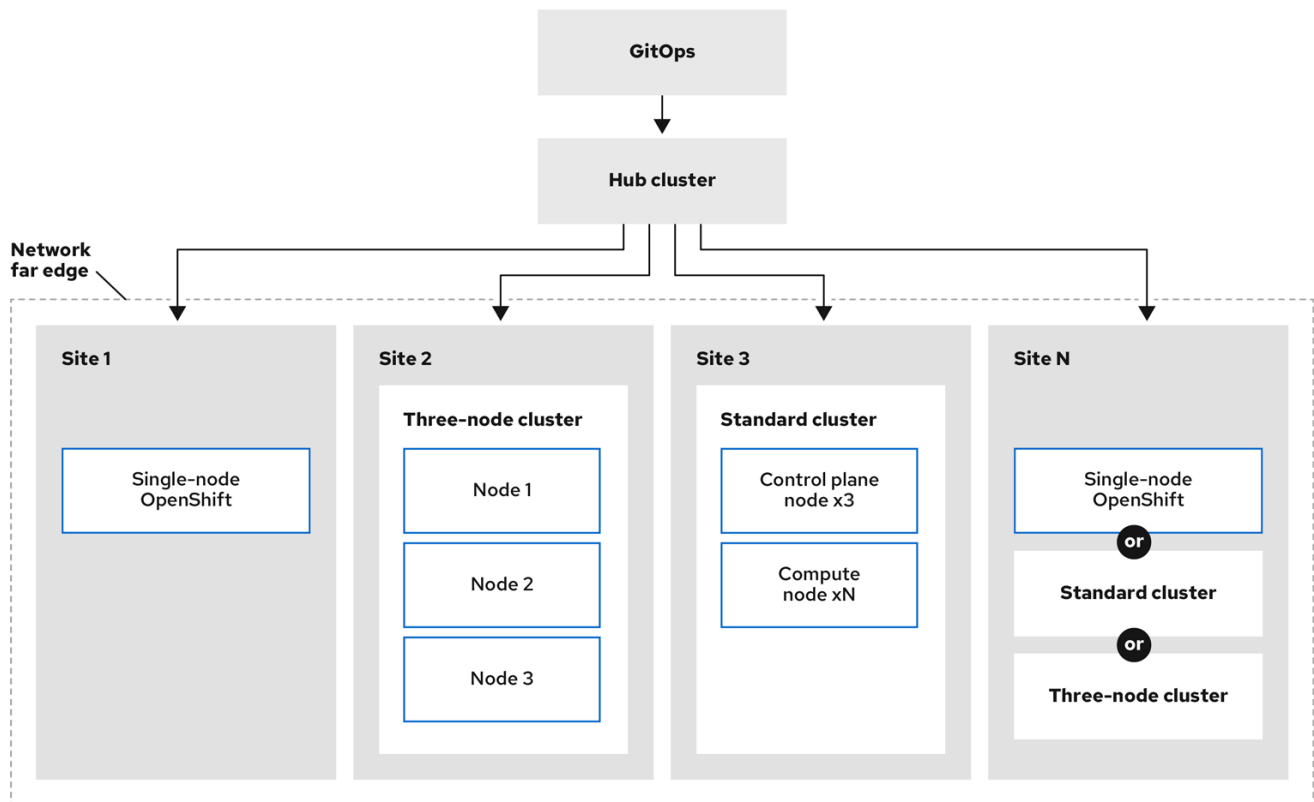
GitOps Zero Touch Provisioning (ZTP) と **GitOps** は、ベアメタル機器の宣言的なサイト定義と設定を使用してリモートエッジサイトを大規模にプロビジョニングできるようにすることで、これらの課題を解決します。テンプレートまたはオーバーレイ設定は、CNF ワークロードに必要な OpenShift Container Platform 機能をインストールします。インストールとアップグレードの全ライフサイクルは、GitOps ZTP パイプラインを通じて処理されます。

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

17.1.2. 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

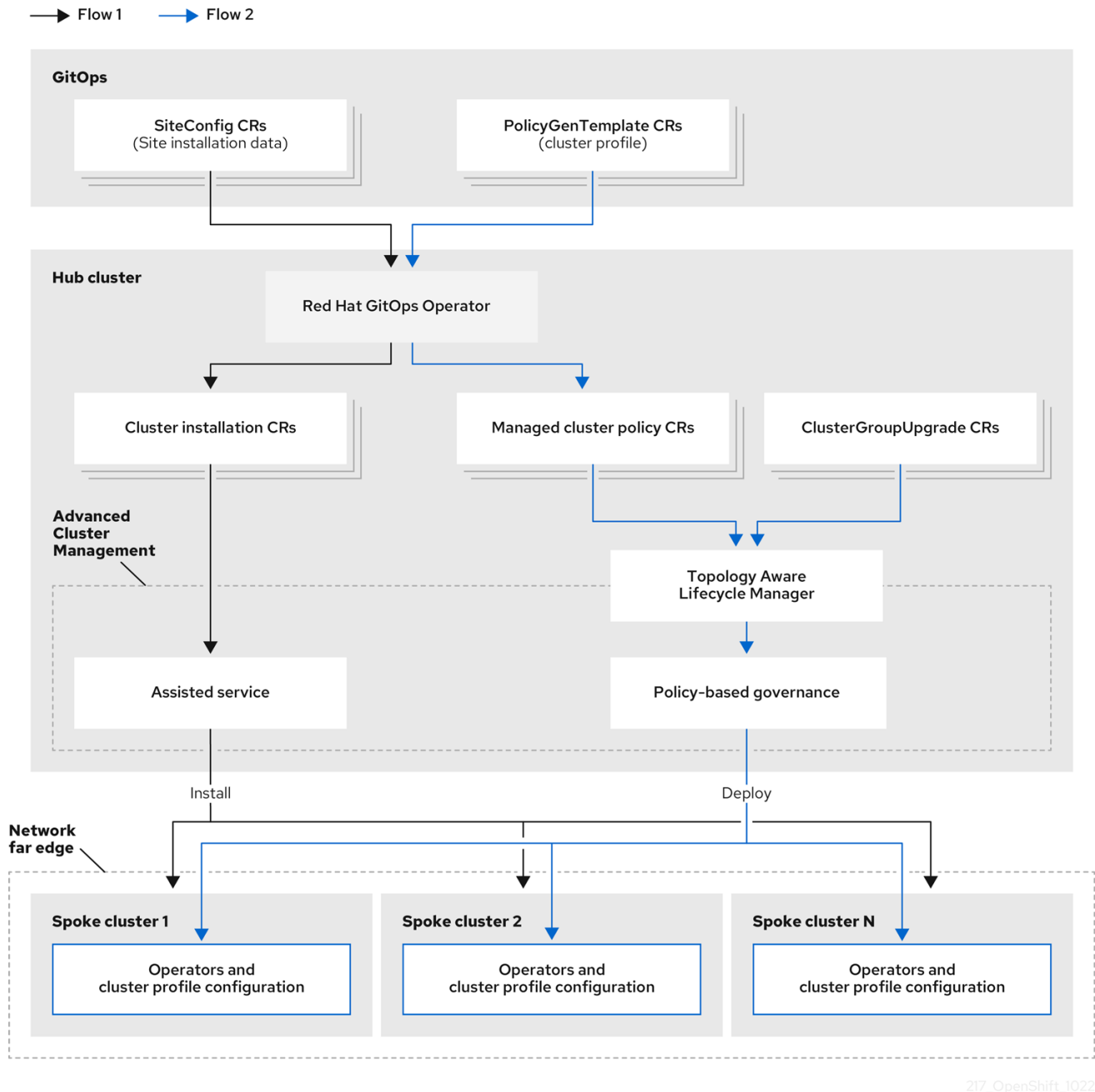
Red Hat Advanced Cluster Management (RHACM) は、単一のハブクラスターが多数のスポーククラスターを管理するハブアンドスポークアーキテクチャーでクラスターを管理します。RHACM を実行するハブクラスターは、GitOps Zero Touch Provisioning (ZTP) と、RHACM のインストール時にデプロイメントされるアシストサービスを使用して、マネージドクラスターのプロビジョニングおよびデプロイメントを実行します。

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。



17.1.3. 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。

17.1.4. 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。

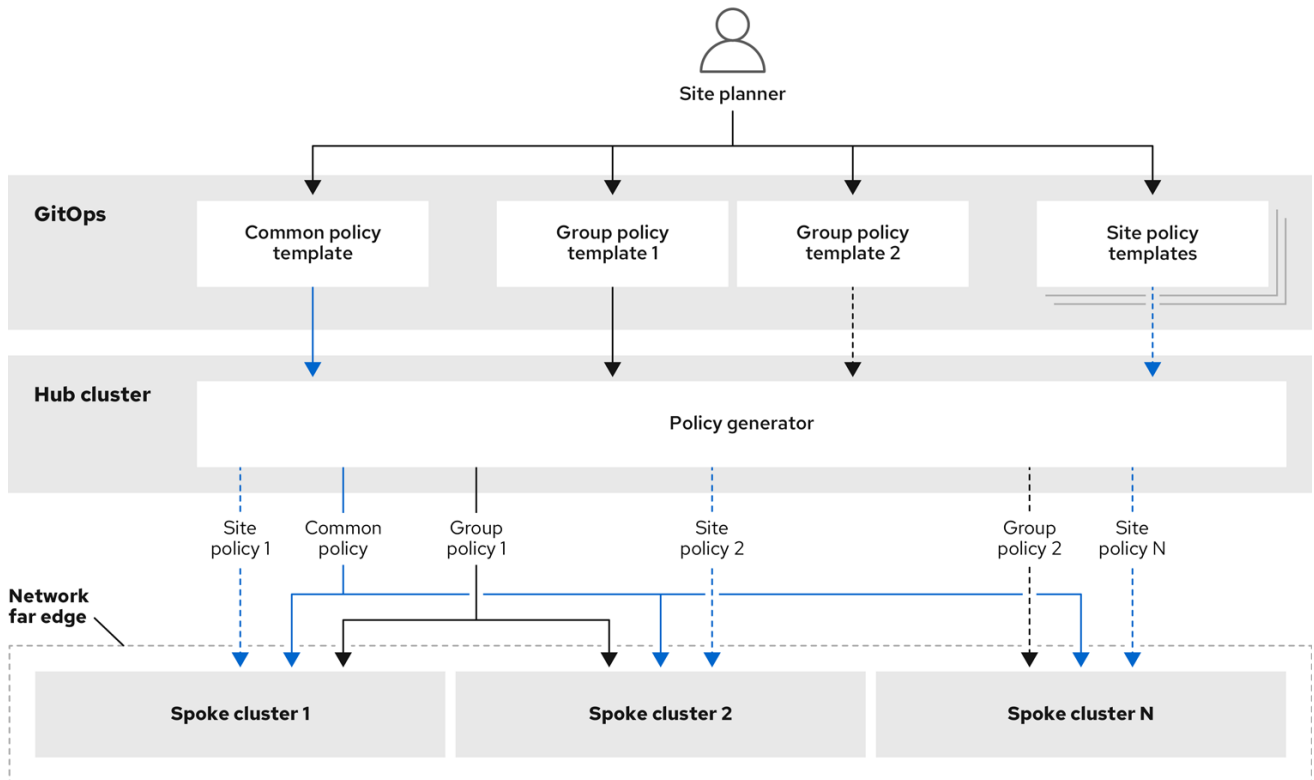


注記

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17.1 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。

17.2. ZTP 用のハブクラスタの準備

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。

17.2.1. Telco RAN 4.13 検証済みソリューションソフトウェアバージョン

Red Hat Telco Radio Access Network (RAN) バージョン 4.13 ソリューションは、次の Red Hat ソフトウェア製品を使用して検証されています。

表17.2 Telco RAN 4.13 検証済みソリューションソフトウェア

Product	ソフトウェアバージョン
Hub クラスタの OpenShift Container Platform のバージョン	4.13
GitOps ZTP プラグイン	4.11、4.12、または 4.13
Red Hat Advanced Cluster Management (RHACM)	2.7
Red Hat OpenShift GitOps	1.9、1.10
Topology Aware Lifecycle Manager (TALM)	4.11、4.12、または 4.13

17.2.2. GitOps ZTP で推奨されるハブクラスタ仕様とマネージドクラスタの制限

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数のスケーリング制限は、ハブクラスタ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。

ハブクラスタのリソース

利用可能なハブクラスタのホストリソース (CPU、メモリー、ストレージ) は、ハブクラスタが管理できるクラスタの数を決定する重要な要素です。ハブクラスタ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の数も多くなります。

ハブクラスタストレージ

ハブクラスタホストのストレージ IOPS 評価と、ハブクラスタホストが NVMe ストレージを使用するかどうかは、ハブクラスタのパフォーマンスと管理できるクラスタ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ルアップされており、これらは管理できるクラスターの数を決定する重要な要素です。

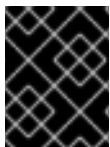
ワークロードのモニタリングと管理

RHACM は、マネージドクラスターを継続的にモニタリングおよび管理します。ハブクラスター上で実行されるモニタリングおよび管理ワークロードの数と複雑さは、ハブクラスターの容量に影響を与える可能性があります。集中的なモニタリングや頻繁な調整操作には追加のリソースが必要となる場合があり、管理可能なクラスターの数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、社内ラボのベンチマークテストのみに基づいており、実際のホストの完全な仕様を表すものではありません。

表17.3 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	バージョン 4.13
RHACM バージョン	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

要件	説明
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17.4 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](#)

17.2.3. 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。
- クラスターで使用するために、切断されたミラーレジストリーを設定しました。



注記

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

手順

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

関連情報

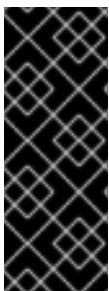
- [OpenShift GitOps のインストール](#)
- [TALM のインストール](#)
- [Operator カタログのミラーリング](#)

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

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

前提条件

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



重要

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

手順

1. ミラーホストにログインします。
2. mirror.openshift.com から RHCOS ISO イメージおよび RootFS イメージを取得します。以下は例になります。
 - a. 必要なイメージ名と OpenShift Container Platform のバージョンを環境変数としてエクスポートします。

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

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

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

- ❶ ISO イメージ名 (例: `rhcos-4.13.1-x86_64-live.x86_64.iso`)
- ❶ RootFS イメージ名 (例: `rhcos-4.13.1-x86_64-live-rootfs.x86_64.img`)
- ❶ OpenShift Container Platform バージョン (例: `4.13.1`)

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/4.13/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.13/${OCP_VERSION}/${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-4.13.1-x86_64-live.x86_64.iso
rhcos-4.13.1-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](#)

17.2.5. 支援サービスの有効化

Red Hat Advanced Cluster Management (RHACM) は、アシストサービスを使用して OpenShift Container Platform クラスターをデプロイします。Red Hat Advanced Cluster Management (RHACM) で MultiClusterHub Operator を有効にすると、支援サービスが自動的にデプロイされます。その後、すべての 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: "4.13"
  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。

17.2.6. 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) 2.8 がインストールされた切断されたハブクラスターのインストールがあります。
- 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 ❶
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 ❷
  osImages:
    - openshiftVersion: <ocp_version>
      url: <iso_url> ❸
```

- ❶ **ConfigMap** namespace と一致するように、**AgentServiceConfig** namespace を **multicluster-engine** に設定します。
- ❷ 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。
- ❸ **httpd** サーバーでホストされる ISO の URL を設定します。



重要

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

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

17.2.7. 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

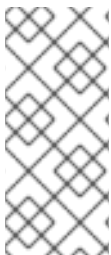
1. 次のコマンドを実行して、**AgentServiceConfig** カスタムリソース (CR) を更新します。

```
$ 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、以下のようになります。

<unauthenticated_registry>

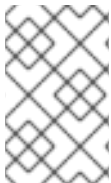
unauthenticated-image-registry.openshift-image-registry.svc:5000 などの新しいレジストリーです。

出力例

```
Login Succeeded!
```

17.2.8. 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 Day1 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には、以前に **out/argocd/deployment/** ディレクトリに抽出され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する必要があります。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。この機能を無効にするには、次のパッチを適用して、このアドオンに関連するハブクラスターとマネージドクラスター Pod を無効にして削除します。

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

4. 以下のコマンドを使用して、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
```

17.2.9. 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。
2. 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** ディレクトリをエクスポートします。

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.13
```

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



```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-
rhel8:v4.13 extract /home/ztp --tar | tar x -C ./out
```

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

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

out/argocd/example のディレクトリー構造は、Git リポジトリの構造およびコンテンツの参照として機能します。この例には、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。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
│   └── ns.yaml
└── siteconfig
    ├── example-sno.yaml
    ├── KlusterletAddonConfigOverride.yaml
    └── kustomization.yaml
```

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

このディレクトリー構造と **kustomization.yaml** ファイルはコミットされ、Git リポジトリにプッシュされる必要があります。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。**SiteConfig** (**example-sno.yaml**) および **PolicyGenTemplate** (**common-ranGen.yaml**、**group-du-sno*.yaml**、および **example-sno-site.yaml**) ファイルは省略され、後でサイトをデプロイする際にプッシュできます。

KlusterletAddonConfigOverride.yaml ファイルは、その CR を参照する1つ以上の **SiteConfig** CR がコミットされ、Git にプッシュされている場合のみ必要です。これがどのように使用されるかについては、**example-sno.yaml** を参照してください。

17.3. 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。

17.3.1. 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。

Waves

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。

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

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

クラスターのデプロイメントには、以下が含まれます。

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

マネージドサイトのインストールプロセスの概要

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

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

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



重要

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

17.3.3. 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。

17.3.4. 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 4.13 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
```

17.3.5. 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必要です。

ネットワーク接続

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

Baseboard Management Controller (BMC) の詳細

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

手順

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


```
$ export CLUSTERNS=example-sno
```
 - b. namespace を作成します。


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



注記

シークレットは、名前が **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 リポジトリーのローカルクローンに、クラスターの **SiteConfig** CR を作成します。
 - a. **out/argocd/example/siteconfig/** フォルダーから CR の適切な例を選択します。フォルダーには、単一ノード、3 ノード、標準クラスターのサンプルファイルが含まれます。
 - **example-sno.yaml**
 - **example-3node.yaml**
 - **example-standard.yaml**
 - b. サンプルファイルのクラスターおよびホスト詳細を、必要なクラスタータイプに一致するように変更します。以下に例を示します。

シングルノード OpenShift SiteConfig CR の例

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
```



```
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  cpuPartitioningMode: AllNodes
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    installConfigOverrides: |
      {
        "capabilities": {
          "baselineCapabilitySet": "None",
          "additionalEnabledCapabilities": [
            "marketplace",
            "NodeTuning"
          ]
        }
      }
    clusterLabels:
      common: true
      group-du-sno: ""
      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
    # crTemplates:
    # KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
    nodes:
      - hostName: "example-node1.example.com"
        role: "master"
        bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbb:1]/redfish/v1/Systems/System.Embedded.1"
        bmcCredentialsName:
          name: "example-node1-bmh-secret"
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        bootMode: "UEFI"
        rootDeviceHints:
          wwn: "0x11111000000asd123"
        # diskPartition:
        # - device: /dev/disk/by-id/wwn-0x11111000000asd123 # match
    rootDeviceHints
      # partitions:
      # - mount_point: /var/imageregistry
```

```
#    size: 102500
#    start: 344844
ignitionConfigOverride: |
{
  "ignition": {
    "version": "3.2.0"
  },
  "storage": {
    "disks": [
      {
        "device": "/dev/disk/by-id/wwn-0x11111000000asd123",
        "wipeTable": false,
        "partitions": [
          {
            "sizeMiB": 16,
            "label": "httpevent1",
            "startMiB": 350000
          },
          {
            "sizeMiB": 16,
            "label": "httpevent2",
            "startMiB": 350016
          }
        ]
      }
    ]
  },
  "filesystem": [
    {
      "device": "/dev/disk/by-partlabel/httpevent1",
      "format": "xfs",
      "wipeFilesystem": true
    },
    {
      "device": "/dev/disk/by-partlabel/httpevent2",
      "format": "xfs",
      "wipeFilesystem": true
    }
  ]
}
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:
          - 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。

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](#)

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

表17.5 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> <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>
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>
spec.clusterImageSettNameRef	<p>サイト内のすべてのクラスターのハブクラスターで使用できるイメージセットを設定します。ハブクラスターでサポートされるバージョンの一覧を表示するには、oc get clusterimagesets を実行します。</p>
installConfigOverrides	<p>クラスターのインストール前に、installConfigOverrides フィールドを設定して、オプションのコンポーネントを有効または無効にします。</p> <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	<p>個々のクラスターをデプロイするために使用されるクラスターイメージセットを指定します。定義されている場合、サイトレベルで spec.clusterImageSetNameRef をオーバーライドします。</p>
spec.clusters.clusterLabels	<p>定義した PolicyGenTemplate CR の bindingRules フィールドに対応するクラスター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ーに適用され、policygentemplates/group-du-sno-ranGen.yaml は group-du-sno: "" が設定されたすべてのクラスターに適用されます。</p>
spec.clusters.crTemplates.KlusterletAddonConfig	<p>オプション: klusterletaddonconfig を、クラスター用に作成された KlusterletAddonConfigOverride.yaml to override the default KlusterletAddonConfig に設定します。</p>
spec.clusters.nodes.hostName	<p>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。</p>

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="491 705 596 837" style="float: left; margin-right: 10px;"> </div> <p style="margin-left: 40px;">注記</p> <p style="margin-left: 40px;">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>

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](#)

17.3.6. 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]'
```

17.3.7. 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
```

17.3.8. 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。

17.3.9. 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 [マネージメントからのクラスターの削除](#) を参照してください。

17.3.10. 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>
```

17.3.11. 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、サイトリポジトリにプッシュします。

17.4. 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

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。

17.4.1. 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 ③
  policyName: "config-policy"
- fileName: DefaultCatsrc.yaml ④
  policyName: "config-policy" ⑤
  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

```

- ① **common: true** は、このラベルを持つすべてのクラスターにポリシーを適用します。
- ② **sourceFiles** の下にリストされているファイルは、インストールされたクラスターの Operator ポリシーを作成します。
- ③ **OperatorHub.yaml** は、切断されたレジストリーの OperatorHub を設定します。
- ④ **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。
- ⑤ **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
      ....

```

17.4.2. 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ルリングに関する推奨事項は、[パフォーマンスおよびスケラビリティ](#) 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。

17.4.3. 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17.6 RAN デプロイメントの PolicyGenTemplate CR

PolicyGenTemplate CR	説明
<code>example-multinode-site.yaml</code>	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。
<code>example-sno-site.yaml</code>	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](#)

17.4.4. 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](#)

17.4.5. 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。

17.4.6. 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。

17.4.7. 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。

17.4.8. 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

- 影響を受ける CR から不要になったコンテンツを削除します。この例では、**SriovOperatorConfig** CR から **disableDrain: false** 行が削除されました。

CR の例:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    "node-role.kubernetes.io/$mcp": ""
```

```

disableDrain: true
enableInjector: true
enableOperatorWebhook: true

```

2. **group-du-sno-ranGen.yaml** ファイル内で、影響を受けるポリシーの **complianceType** を **mustonlyhave** に変更します。

サンプル YAML

```

# ...
- fileName: SriovOperatorConfig.yaml
  policyName: "config-policy"
  complianceType: mustonlyhave
# ...

```

3. **ClusterGroupUpdates** CR を作成し、CR の変更を受け取る必要があるクラスターを指定します。

ClusterGroupUpdates CR の例

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-remove
  namespace: default
spec:
  managedPolicies:
    - ztp-group.group-du-sno-config-policy
  enable: false
  clusters:
    - spoke1
    - spoke2
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
  batchTimeoutAction:

```

4. 以下のコマンドを実行して **ClusterGroupUpgrade** CR を作成します。

```
$ oc create -f cgu-remove.yaml
```

5. たとえば適切なメンテナンス期間中などに変更を適用する準備が完了したら、次のコマンドを実行して **spec.enable** フィールドの値を **true** に変更します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-remove \
--patch '{"spec":{"enable":true}}' --type=merge
```

検証

1. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get <kind> <changed_cr_name>
```

出力例

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

ポリシーの **COMPLIANCE STATE** が **Compliant** の場合、CR が更新され、不要なコンテンツが削除されたことを意味します。

- 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。

17.4.9. 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。

17.5. 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。

17.5.1. GitOps ZTP インストール CR と設定 CR の手動生成

ztp-site-generate コンテナの **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
```

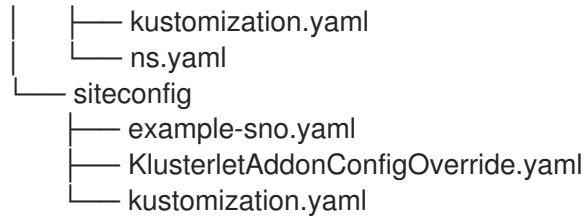
2. **ztp-site-generate** コンテナイメージから **argocd** ディレクトリーをエクスポートします。

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.13 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"
  cpuPartitioningMode: AllNodes
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.10"
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
  - clusterName: "example-sno"
    networkType: "OVNKubernetes"
    installConfigOverrides: |
      {
        "capabilities": {
          "baselineCapabilitySet": "None",
          "additionalEnabledCapabilities": [
            "marketplace",
            "NodeTuning"
          ]
        }
      }
  clusterLabels:
    common: true
    group-du-sno: ""
    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
# crTemplates:
# KlusterletAddonConfig: "KlusterletAddonConfigOverride.yaml"
nodes:
- hostName: "example-node1.example.com"
  role: "master"
  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"
    bootMode: "UEFI"
    rootDeviceHints:
      wwn: "0x11111000000asd123"
# diskPartition:
# - device: /dev/disk/by-id/wwn-0x11111000000asd123 # match rootDeviceHints
# partitions:
# - mount_point: /var/imageregistry
# size: 102500
# start: 344844
ignitionConfigOverride: |
{
  "ignition": {
    "version": "3.2.0"
  },
  "storage": {
    "disks": [
      {
        "device": "/dev/disk/by-id/wwn-0x11111000000asd123",
        "wipeTable": false,
        "partitions": [
          {
            "sizeMiB": 16,
            "label": "httpevent1",
            "startMiB": 350000
          },
          {
            "sizeMiB": 16,
            "label": "httpevent2",
            "startMiB": 350016
          }
        ]
      }
    ]
  },
  "filesystem": [
    {
      "device": "/dev/disk/by-partlabel/httpevent1",
      "format": "xfs",
      "wipeFilesystem": true
    },
    {
      "device": "/dev/disk/by-partlabel/httpevent2",
      "format": "xfs",
      "wipeFilesystem": true
    }
  ]
}
```

```

    ]
  }
}
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:
            - 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

```

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

```

$ podman run -it --rm -v `pwd`/out/argocd/example/siteconfig:/resources:Z -v `pwd`/site-install:/output:Z,U registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.13 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:v4.13 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:v4.13 generator config -N . /output
```

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

出力例

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

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

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](#)

17.5.2. マネージドベアメタルホストシークレットの作成

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



注記

シークレットは、**SiteConfig** CR から名前参照されます。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

```

- 1 関連する **SiteConfig** CR で設定された namespace と一致する必要があります
- 2 **password** と **username** の Base64 エンコード値
- 3 関連する **SiteConfig** CR で設定された namespace と一致する必要があります
- 4 Base64 でエンコードされたプルシークレット

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

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

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



注記

OpenShift Container Platform 4.13 では、カーネル引数の追加のみを行うことができます。カーネル引数を置き換えたり削除したりすることはできません。

前提条件

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

手順

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

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

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



注記

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

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
```

17.5.4. 単一のマネージドクラスターのインストール

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

前提条件

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

手順

1. デプロイする特定のクラスターバージョンごとに **ClusterImageSet** を作成します (例: **clusterImageSet-4.13.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-4.13.0 1
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.13.0-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-4.13.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](#)

17.5.5. マネージドクラスターのインストールステータスの監視

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

前提条件

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

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

True はマネージドクラスターの準備が整っていることを示します。

2. エージェントのステータスを確認します。

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

3. **describe** コマンドを使用して、エージェントの条件に関する詳細な説明を指定します。認識できるステータスには、**BackendError**、**InputError**、**ValidationsFailing**、**InstallationFailed**、および **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
```

17.5.6. マネージドクラスターのトラブルシューティング

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

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

出力例

NAME	HUB ACCEPTED	MANAGED CLUSTER	URLS	JOINED	AVAILABLE
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
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。

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

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



注記

すべてのマネージドクラスターには独自の namespace があり、**ManagedCluster** と **ClusterImageSet** を除くすべてのインストール CR はその namespace の下にありません。**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17.7 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。
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。
Managed Cluster	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。

CR	説明	使用法
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。

17.6. 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

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

関連情報

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

17.6.1. OpenShift Container Platform で低レイテンシーのアプリケーションを実行する

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

RHCOS のリアルタイムカーネル

ワークロードが高レベルのプロセス決定で処理されるようにします。

CPU の分離

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

NUMA 対応のトポロジー管理

メモリーと Huge Page を CPU および PCI デバイスに合わせて、保証されたコンテナメモリーと 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。

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

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

表17.8 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。

- (コアあたりのスレッド数×コア)×ソケット=vCPU



重要

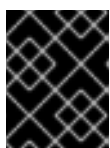
仮想メディアを使用して起動する場合は、サーバーには Baseboard Management Controller (BMC) が必要です。

17.6.3. 低遅延と高パフォーマンスのためのホストファームウェアの設定

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

手順

1. UEFI/BIOS Boot Mode を **UEFI** に設定します。
2. ホスト起動シーケンスの順序で、**ハードドライブ** を設定します。
3. ハードウェアに特定のファームウェア設定を適用します。以下の表は、Intel FlexRAN 4G および 5G baseband PHY 参照設計をベースとした Intel Xeon Skylake 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17.9 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
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。

17.6.4. マネージドクラスターネットワークの接続の前提条件

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

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

- `api.sno-managed-cluster-1.internal.domain.com`
- `console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com`

17.6.5. GitOps ZTP を使用した単一ノードの OpenShift でのワークロードの分割

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

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

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



注記

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

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

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



重要

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

関連情報

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

17.6.6. 推奨されるクラスターインストールマニフェスト

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



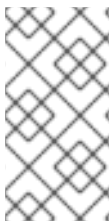
注記

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

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

17.6.6.1. ワークロードの分割

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



注記

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



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、クラスターのプロビジョニングに使用する **/extra-manifest** フォルダーからワークロードパーティショニングの **MachineConfig** CR を削除します。

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

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

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

検証

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

1. マネージドクラスターへのリモートシェルプロンプトを開きます。

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

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


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

出力例

```
pid 8481's current affinity list: 0-1,52-53
pid 8726's current affinity list: 0-1,52-53
pid 9088's current affinity list: 0-1,52-53
pid 9945's current affinity list: 0-1,52-53
pid 10387's current affinity list: 0-1,52-53
pid 12123's current affinity list: 0-1,52-53
pid 13313's current affinity list: 0-1,52-53
```

- システムアプリケーションの CPU ピニングが正しいことを確認します。

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

出力例

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

17.6.6.2. プラットフォーム管理フットプリントの削減

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

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

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,IyEvYmluL2Jhc2gKCmRIYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkgewogIGVjaG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDApIFVOSVQgW2VudmZpbGUgW3Zhcml5bWVudXQogIGVjaG8gKICBIY2hviEV4dHJhY3QgdGhllGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3Rhcnc3RhbnphIGZyb20gdGhllGdpdmVulHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJlIGl0IHRvIHNoZG91dAogIGVj
```

```
aG8KICBIY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQzIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZW
FkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGV
mYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUIGlmlG5vdCBzcGVjaWZpZWQzIiAgZlZhdCAXC
n0KClVOSVQ9JDEKRU5WRkIMRT0kMgpWQVJOQU1FPSQzCmlmIFt1C16ICRVTKlUIHx8ICRVTKlUI
D09ICItLWlhbHAiHx8ICRVTKlUID09ICItaCgXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cm
FjdGluZyBFFeGVjU3RhcncgZnJvbSAkVU5JVCIKRkIMRT0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCB
oZWFKIC1uIDEpCkZJTEU9JHtGSUxFl1wjlH0KaWYgW1sglSAtZiAkRkIMRSBdXTsgdGhlgogIGRIYnV
nICJGYWlsZWQgdG8gZmluZCBYb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSikICBle
GI0CmZpCmRIYnVnICJTZXJ2aWNlIGRIZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0k
KHNIZCAtbiAtZSAnL15FeGVjU3RhcncgQ9LipXCQvLC9bXlxcXSQvIHsgcy9eRXh1Y1N0YXJ0PS8vOyBw
IH0nIC1lICcvXkV4ZWNTdGFydD0uKlIeXFxdJC8geyBzL15FeGVjU3RhcncgQ9Ly87IHAgfScgJEZJTEUp
CgppZiBbWyAkRU5WRkIMRSBdXTsgdGhlgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUI
R9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkJTEUKZWxzZQogIGVja
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

```

17.6.6.3. SCTP

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

推奨される SCTP 設定 (03-sctp-machine-config-master.yaml)

```

# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module-master
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf

```

17.6.6.4. コンテナの起動の高速化


```
tc3RhdGUgcG9kIGNvdW50CiAgICBjY291bnQ9JChjcmldGwgchMgfCB3YyAtbCkKICAgIGlmIHN0ZW
FkeXN0YXRlICRsYXN0Q2NvdW50ICRjY291bnQ7IHRoZW4KICAgICAgKChzdGVhZlITdGF0ZVRpb
WUgKz0gcykpcCiAgICAgIGVjaG8gIlnN0ZWFkeS1zdGF0ZSBmb3IglJHtZdGVhZlITdGF0ZVRpbWV9cy8
ke1NURUFEWV9TVEFURV9XSU5ET1d9cyIKICAgICAgAgaWYgW1sgJHN0ZWFkeVN0YXRIVGltZSAZ2Z
UgJFNURUFEWV9TVEFURV9XSU5ET1cgXV07IHRoZW4KICAgICAgICBs2dnZXIglJiY292ZXJ5Oib
TdGVhZHktdG9kUgKCsLSAqU1RFQURZX1NUQVRFX1RIUkVTSE9MRCkgZm9yICR7U1RFQU
RZX1NUQVRFX1dJTkRlV31zOiBEb25lIlgogICAgICAgIHJldHVybiAwCiAgICAgIGZpCiAgICBlbHNICiAg
ICAgIGlmIFtIbCRzdGVhZlITdGF0ZVRpbWUgLVd0IDAgXV07IHRoZW4KICAgICAgICBIY2hvlCJSZX
NldHRpbmcmcg3RIYWR5LXN0YXRlIHRpbWVylgogICAgICAgIHN0ZWFkeVN0YXRIVGltZT0wCiAgICAgI
GZpCiAgICBlbmaQogICAgbGFzdENjbn3VudD0kY2NvdW50CiAgZG9uZQogIGxvZ2dlciAiUmVjb3Zlcnk6I
FJlY292ZXJ5IENvbXBsZXRlIFRpbWVvdXQiCn0KCm1haW4oKSB7CiAgAgaWYglSB1bnJlc3RyaWN0ZW
RDChVzZXQgPiYvZGV2L251bGw7IHRoZW4KICAgIGxvZ2dlciAiUmVjb3Zlcnk6IE5vIHVuZmVzdHJpY3
RIZCBDCkVzZXQgY291bGQgYmUgZGV0ZWN0ZWQicAgICBjYXR1cm4gMQogIGZpCgogIGlmICE
gcmVzdHJpY3RIZENwdXNldCA+Ji9kZXlYbnVsbDsgdGhIbGogICAgbG9nZ2VlIjCJSZWNvdmVyeTogT
m8gcmVzdHJpY3RIZCBDCkVzZXQgaGFzIGJlZ2V4Y29uZmlndXJlZC4gIkdllGfYzSBhbHJlYWR5IHJ1
bm5pbmcmgdW5yXN0cmldGVkLilKICAgIHJldHVybiAwCiAgZmkkKiAglyBFbnN1cmUgd2UgcmVzZXQg
dGhIENQVSbHmZpbml0eSB3aGVuIHdldGV4aXQgdGhpcyBzY3JpcHQgZm9yIGFueSBYzWfzb24KI
CAjIFRoaxMgd2F5IGVpdGhIciBhZnRlciB0aGUgdGltZXIglZlZlZyBvciBhZnRlciB0aGUgcHJvY2Vzc
yBpcyBpbmRlcnJ1cHRlZAogICMgdmlhIF5DIG9yIFNJR1RFUk0sIHdIHJldHVybiB0aGluZ3MgYmFjayB0
byB0aGUgd2F5IHRoZXkgc2hvdWxkIGJlLgogIHRYAgc2V0UmVzdHJpY3RIZCBFWEIUCgogIGxvZ2
dlciAiUmVjb3Zlcnk6IFJlY292ZXJ5IE1vZGUgU3RhcncRpbmciCiAgc2V0VW5yXN0cmldGVkCiAgd2Fpd
EZvclJlYWR5Cn0KCmImIFtIbCRzdGVhZlITdGF0ZVRpbWUgLVd0IDAgXV07IHRoZW4KICBtYV
luIClke0B9IlgogIGV4aXQgJD8KZmkK
```

```
mode: 493
```

```
path: /usr/local/bin/accelerated-container-startup.sh
```

```
systemd:
```

```
units:
```

```
- contents: |
```

```
[Unit]
```

```
Description=Unlocks more CPUs for critical system processes during container startup
```

```
[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 = 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: 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

```

17.6.6.5. kdump による自動カーネルクラッシュダンプ

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

ice ドライバーを削除するための推奨 MachineConfig (05-kdump-config-master.yaml)

```

# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 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,lyEvdXNyL2Jpbi9lbnYgYmFzaAoKlyBUaGlzIHJncmlwdCByZW1vdmVzIHRoZSBpY2UgbW9k
dWxlIGZyb20ga2R1bXAgdG8gcHJldmVudCBvZHVtcCBmYWlscXJlcyBvbiBjZXJ0YWluIHJlcnZlcnMuCi
MgVGhpcyBpcyBhIHRlbnVcmFyeSB3b3JrYXJvdW5kIGZvcjBSSEVMUExBT0xMzgyMzYgYW5kIGNh
iBiZSByZW1vdmVkiHdoZW4gdGhhdCBpc3N1ZSBpcwojIGZpeGVkLgoKc2V0IC14CgpTRUQ9li91c3lv
YmluL3NIZCIKR1JFUD0iL3Vzci9iaW4vZ3JlcCIKCiMgb3ZlcnJpZGUgZm9yIHRlc3RpbmcmcG9zZX
MKS0RVTVBfQ09ORj0iJHsxOi0vZXRjL3N5c2NvbmZpZy9rZHVtcH0iClJFTU9WRV9JQ0VfU1RSPSJtb
2R1bGVfYmxhY2tsaXN0PWljZSIKCiMgZXhpdCBpZiBmaWxlIGRvZXNuJ3QgZXhpc3QKWyAhIC1mIC
R7S0RVTVBfQ09ORn0gXSAmJiBleGl0IDAKCiMgZXhpdCBpZiBmaWxlIGFscmVhZHKgdXBkYXRIZAAok
e0dSRVB9IC1GcSAke1JFTU9WRV9JQ0VfU1RSfSAke0tEVU1QX0NPTkZ9ICYmIGV4aXQgMAoKlyB
UYXJnZXQgbGluZSBsb29rcyBzb21ldGhpbmcmcGlrZSB0aGlzOgojIEtEVU1QX0NPTU1BTkRMSU5FX
0FQUEVORD0iaXJxcG9sbCBucj09cHVzPTEgLi4uIGhlc3RfZGZlYWJsZSIKlyBVc2Ugc2VklHRvIG1hdG
NoIGV2ZXJ5dGhpbmcmcYmV0d2VlbiB0aGUgcXVvdGVzIGFuZCBhcHBlbmQgdGhlfJFTU9WRV9JQ0
VfU1RSIHRvIGl0CiR7U0VEfSAAtaSAncy9eS0RVTVBfQ09NTUFORExJTkVfQVBQRU5EPSJbXiJdKi8m
ICcke1JFTU9WRV9JQ0VfU1RSfScvJyAke0tEVU1QX0NPTkZ9IHx8IGV4aXQgMAo=
          mode: 448
          path: /usr/local/bin/kdump-remove-ice-module.sh

```

推奨される kdump 設定 (06-kdump-master.yaml)

```

# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 06-kdump-enable-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: kdump.service
    kernelArguments:
      - crashkernel=512M

```


17.6.6.6. CRI-O キャッシュの自動ワイブを無効にする

制御されていないホストのシャットダウンまたはクラスターの再起動の後、CRI-O は CRI-O キャッシュ全体を自動的に削除します。そのため、ノードの再起動時にはすべてのイメージがレジストリーからプルされます。これにより、許容できないほど復元に時間がかかったり、復元が失敗したりする可能性があります。GitOps ZTP を使用してインストールするシングルノード OpenShift クラスターでこの問題が発生しないようにするには、クラスターをインストールする際に CRI-O 削除キャッシュ機能を無効にします。

コントロールプレーンノードで CRI-O キャッシュワイブを無効にするための推奨 MachineConfig CR (99-crio-disable-wipe-master.yaml)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-crio-disable-wipe-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAilgo=
            mode: 420
            path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

ワーカーノードで CRI-O キャッシュワイブを無効にするための推奨 MachineConfig CR (99-crio-disable-wipe-worker.yaml)

```
# Automatically generated by extra-manifests-builder
# Do not make changes directly.
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-crio-disable-wipe-worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,W2NyaW9dCmNsZWFuX3NodXRkb3duX2ZpbGUgPSAilgo=
            mode: 420
            path: /etc/crio/crio.conf.d/99-crio-disable-wipe.toml
```

17.6.6.7. 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 CR (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 CR (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
```

17.6.7. 推奨されるインストール後のクラスター設定

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



注記

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

17.6.7.1. Operator namespace と Operator グループ

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、以下の **OperatorGroup** および **Namespace** カスタムリソース (CR) が必要です。

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

次の CR が必要です。

推奨される Storage Operator Namespace と OperatorGroup 設定

```
---
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
spec:
  targetNamespaces:
  - openshift-local-storage
```

推奨される Cluster Logging Operator Namespace と OperatorGroup 設定

```
---
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
spec:
  targetNamespaces:
    - openshift-logging

```

推奨される PTP Operator Namespace と OperatorGroup 設定

```

---
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-ntp
spec:
  targetNamespaces:
    - openshift-ntp

```

推奨される SR-IOV Operator Namespace と OperatorGroup 設定

```

---
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
spec:
  targetNamespaces:
    - openshift-sriov-network-operator

```

17.6.7.2. Operator のサブスクリプション

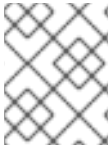
DU ワークロードを実行する単一ノードの OpenShift クラスターには、次の **Subscription** CR が必要です。サブスクリプションは、次の Operator をダウンロードする場所を提供します。

- Local Storage Operator

- Logging Operator
- PTP Operator
- SR-IOV Network Operator

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。



注記

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、計画/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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: "stable"
  name: local-storage-operator
  source: redhat-operators
  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

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

推奨される PTP Operator サブスクリプション

```
---
apiVersion: operators.coreos.com/v1alpha1
```

```

kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  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

```

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

```

17.6.7.3. クラスターのロギングとログ転送

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

推奨されるクラスターログとログ転送の設定

```

apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  managementState: "Managed"
  curation:
    type: "curator"
    curator:
      schedule: "30 3 * * *"
  collection:
    logs:
      type: "fluentd"
      fluentd: {}

```

推奨されるログ転送設定

```

apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
    - type: "kafka"
      name: kafka-open
      url: tcp://10.46.55.190:9092/test
  inputs:
    - name: infra-logs
      infrastructure: {}
  pipelines:
    - name: audit-logs
      inputRefs:
        - audit
      outputRefs:
        - kafka-open
    - name: infrastructure-logs
      inputRefs:
        - infrastructure
      outputRefs:
        - kafka-open

```

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

17.6.7.4. パフォーマンスプロファイル

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



注記

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

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

推奨されるパフォーマンスプロファイル設定

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
  additionalKernelArgs:
    - "rcupdate.rcu_normal_after_boot=0"
    - "efi=runtime"
    - "module_blacklist=irdma"
  cpu:

```

```

isolated: 2-51,54-103
reserved: 0-1,52-53
hugepages:
  defaultHugepagesSize: 1G
pages:
  - count: 32
    size: 1G
    node: 0
machineConfigPoolSelector:
  pools.operator.machineconfiguration.openshift.io/master: ""
nodeSelector:
  node-role.kubernetes.io/master: "
numa:
  topologyPolicy: "restricted"
realTimeKernel:
  enabled: true
workloadHints:
  realTime: true
  highPowerConsumption: false
  perPodPowerManagement: false

```

表17.10 シングルノード OpenShift クラスターの PerformanceProfile CR オプション

PerformanceProfile CR フィールド	説明
metadata.name	<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>

PerformanceProfile CR フィールド	説明
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 style="display: flex; align-items: flex-start;">  <div> <p>重要</p> <p>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> </div> </div>
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>
spec.workloadHints	<p>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。</p>

17.6.7.5. PTP

単一ノードの OpenShift クラスターは、ネットワーク時間同期に Precision Time Protocol (PTP) を使用します。次の **PtpConfig** CR の例は、必要な PTP スレーブ設定を示しています。

推奨される PTP 設定

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: slave
  namespace: openshift-ptp
spec:
  profile:
    - name: "slave"

```

```
# The interface name is hardware-specific
interface: ens5f0
ptp4lOpts: "-2 -s"
phc2sysOpts: "-a -r -n 24"
ptpSchedulingPolicy: SCHED_FIFO
ptpSchedulingPriority: 10
ptpSettings:
  logReduce: "true"
ptp4lConf: |
[global]
#
# Default Data Set
#
twoStepFlag 1
slaveOnly 0
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
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0xA0
recommend:
- profile: "slave"
```

```
priority: 4
match:
- nodeLabel: "node-role.kubernetes.io/master"
```

17.6.7.6. 拡張調整済みプロファイル

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

推奨される拡張 Tuned プロファイル設定

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - name: performance-patch
      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: "master"
    priority: 19
    profile: performance-patch
```

表17.11 シングルノード OpenShift クラスター用に調整された 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-<code>#{PerformanceProfile.metadata.name}</code> 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。

17.6.7.7. 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 設定

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    "node-role.kubernetes.io/master": ""
  enableInjector: true
  enableOperatorWebhook: true
```

表17.12 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 SrioNetwork 設定

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetwork
metadata:
  name: ""
  namespace: openshift-sriov-network-operator
spec:
  resourceName: "du_mh"
  networkNamespace: openshift-sriov-network-operator
  vlan: "150"
  spoofChk: ""
  ipam: ""
  linkState: ""
  maxTxRate: ""
  minTxRate: ""
  vlanQoS: ""
  trust: ""
  capabilities: ""

```

表17.13 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
<code>spec.vlan</code>	<code>vlan</code> 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 設定

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetworkNodePolicy
metadata:
  name: $name
  namespace: openshift-sriov-network-operator
spec:
  # Attributes for Mellanox/Intel based NICs
  deviceType: netdevice/vfio-pci
  isRdma: true/false
  nicSelector:
    # The exact physical function name must match the hardware used
    pfNames: [ens7f0]
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8
  priority: 10
  resourceName: du_mh

```

表17.14 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
--------------------------------	----

SriovNetworkNodePolicy CR フィールド	説明
spec.deviceType	deviceType を、 vfio-pci または netdevice として設定します。
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。

17.6.7.8. 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\"}}"
```

17.6.7.9. 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:
    ran.openshift.io/ztp-deploy-wave: "1"
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
    prometheusK8s:
      retention: 24h
```

17.6.7.10. 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

```

17.6.7.11. 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17.15 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。

17.6.7.12. 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
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](#)

17.7. 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

- 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。

17.7.1. vDU クラスターホストの推奨ファームウェア設定

OpenShift Container Platform 4.13 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17.16 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。

ファームウェア設定	設定	説明
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。
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。

ファームウェア設定	設定	説明
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。

17.7.2. vDU アプリケーションを実行するための推奨クラスター設定

仮想化分散ユニット (vDU) アプリケーションを実行するクラスターには、高度に調整かつ最適化された設定が必要です。以下の情報では、OpenShift Container Platform 4.13 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。

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

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

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

表17.17 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	

MachineConfig CR	説明
02-workload-partitioning.yaml	<p>クラスターのワークロードパーティショニングを設定します。クラスターをインストールするときに、この MachineConfig CR を適用します。</p> <p>注記</p> <p>SiteConfig CR の cpuPartitioningMode フィールド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場合、02-workload-partitioning.yaml CR を使用する必要はありません。cpuPartitioningMode フィールドの使用は、OpenShift Container Platform 4.13 のテクノロジープレビュー機能です。詳細は、「GitOps ZTP を使用したシングルノード OpenShift でのワークロードパーティショニング」を参照してください。</p>
03-sctp-machine-config-master.yaml 03-sctp-machine-config-worker.yaml	<p>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。</p>
04-accelerated-container-startup-master.yaml 04-accelerated-container-startup-worker.yaml	<p>クラスターの高速スタートアップを設定します。</p>
05-kdump-config-master.yaml 05-kdump-config-worker.yaml 06-kdump-master.yaml 06-kdump-worker.yaml	<p>クラスターの kdump クラッシュレポートを設定します。</p>
99-crio-disable-wipe-master.yaml 99-crio-disable-wipe-worker.yaml	<p>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。</p>

関連情報

- [ztp-site-generate コンテナからのソース CR の抽出](#)

17.7.2.2. 推奨されるクラスター Operator

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

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

17.7.2.3. 推奨されるクラスターカーネル設定

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

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

```
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-ptp=0:f:10:*:ice-ptp.*
      group.ice-gnss=0:f:10:*:ice-gnss.*
      [service]
      service.stalld=start,enable
      service.chrond=stop,disable
```

17.7.2.4. リアルタイムカーネルバージョンの確認

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

前提条件

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

3. 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。

1. クラスターノードへのリモートシェル接続を開きます。

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

2. リアルタイムカーネルバージョンを確認します。

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

出力例

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

17.7.3. 推奨されるクラスター設定が適用されていることの確認

クラスターが正しい設定で実行されていることを確認できます。以下の手順では、DU アプリケーションを OpenShift Container Platform 4.13 クラスターにデプロイするために必要なさまざまな設定を確認する方法について説明します。

前提条件

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

手順

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

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

出力例

```
spec:
  disableAllDefaultSources: true
```

2. 次のコマンドを実行して、必要なすべての **CatalogSource** リソースにワークロードのパーティショニング (**PreferredDuringScheduling**) のアノテーションが付けられていることを確認します。

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

出力例

```
certified-operators -- {"effect": "PreferredDuringScheduling"}
community-operators -- {"effect": "PreferredDuringScheduling"}
ran-operators 1
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

- 1** アノテーションが付けられていない **CatalogSource** リソースも返されます。この例では、**ran-operators CatalogSource** リソースにはアノテーションが付けられておらず、**PreferredDuringScheduling** アノテーションがありません。



注記

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

3. 該当するすべての OpenShift Container Platform Operator の namespace がワークロードのパーティショニング用にアノテーションされていることを確認します。これには、コア OpenShift Container Platform とともにインストールされたすべての Operator と、参照 DU チューニング設定に含まれる追加の Operator のセットが含まれます。以下のコマンドを実行します。

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

出力例

```
default --
openshift-apiserver -- management
openshift-apiserver-operator -- management
openshift-authentication -- management
openshift-authentication-operator -- management
```



重要

追加の Operator は、ワークロードパーティショニングのためにアノテーションを付けてはなりません。前のコマンドからの出力では、追加の Operator が -- セパレーターの右側に値なしでリストされている必要があります。

4. **ClusterLogging** 設定が正しいことを確認してください。以下のコマンドを実行します。
 - a. 適切な入力ログと出力ログが設定されていることを確認します。

```
$ oc get -n openshift-logging ClusterLogForwarder instance -o yaml
```

出力例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  creationTimestamp: "2022-07-19T21:51:41Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "1030342"
  uid: 8c1a842d-80c5-447a-9150-40350bdf40f0
spec:
  inputs:
  - infrastructure: {}
    name: infra-logs
  outputs:
  - name: kafka-open
    type: kafka
    url: tcp://10.46.55.190:9092/test
```



```

pipelines:
- inputRefs:
  - audit
  name: audit-logs
  outputRefs:
  - kafka-open
- inputRefs:
  - infrastructure
  name: infrastructure-logs
  outputRefs:
  - kafka-open
...

```

- b. キュレーションスケジュールがアプリケーションに適していることを確認します。

```
$ oc get -n openshift-logging clusterloggings.logging.openshift.io instance -o yaml
```

出力例

```

apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  creationTimestamp: "2022-07-07T18:22:56Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "235796"
  uid: ef67b9b8-0e65-4a10-88ff-ec06922ea796
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    curator:
      schedule: 30 3 * * *
      type: curator
  managementState: Managed
...

```

5. 次のコマンドを実行して、Web コンソールが無効になっている (**managementState: Removed**) ことを確認します。

```
$ oc get consoles.operator.openshift.io cluster -o jsonpath="{ .spec.managementState }"
```

出力例

```
Removed
```

6. 次のコマンドを実行して、クラスターノードで **chronyd** が無効になっていることを確認します。

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

ノードで **chronyd** のステータスを確認します。

```
sh-4.4# chroot /host
```

```
sh-4.4# systemctl status chronyd
```

出力例

```
• chronyd.service - NTP client/server
  Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset:
  enabled)
  Active: inactive (dead)
  Docs: man:chronyd(8)
       man:chrony.conf(5)
```

7. **linuxptp-daemon** コンテナへのリモートシェル接続と PTP Management Client (**pmc**) ツールを使用して、PTP インターフェイスがプライマリークロックに正常に同期されていることを確認します。

- a. 次のコマンドを実行して、**\$PTP_POD_NAME** 変数に **linuxptp-daemon** Pod の名前を設定します。

```
$ PTP_POD_NAME=$(oc get pods -n openshift-ptp -l app=linuxptp-daemon -o name)
```

- b. 次のコマンドを実行して、PTP デバイスの同期ステータスを確認します。

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

出力例

```
sending: GET PORT_DATA_SET
3cecef.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 ❶
ingress_time       1657275432697400530
cumulativeScaledRateOffset +0.000000000
scaledLastGmPhaseChange  0
gmTimeBaseIndicator  0
lastGmPhaseChange  0x0000'0000000000000000.0000
gmPresent          true ❷
gmIdentity         3c2c30.fff.670e00
```

- ❶ **master_offset** は -100 から 100 ns の間である必要があります。
- ❷ 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 ❶
        [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

```

❶ `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。

17.8. 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。

17.8.1. GitOps ZTP パイプラインでの追加インストールマニフェストのカスタマイズ

GitOps Zero Touch Provisioning (ZTP) パイプラインのインストールフェーズに追加するマニフェストセットを定義できます。これらのマニフェストは **SiteConfig** カスタムリソース (CR) にリンクされ、インストール時にクラスターに適用されます。インストール時に **MachineConfig** CR を含めると、インストール作業が効率的になります。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリを作成している。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、追加のマニフェスト CR のセットを作成します。
2. カスタムの `/siteconfig` ディレクトリーで、追加のマニフェストの `/extra-manifest` ディレクトリーを作成します。以下の例は、`/extra-manifest` フォルダを持つ `/siteconfig` のサンプルを示しています。

```

siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
├── extra-manifest
│   └── 01-example-machine-config.yaml

```

3. カスタムの追加マニフェスト CR を `siteconfig/extra-manifest` ディレクトリーに追加します。
4. **SiteConfig** CR の `extraManifestPath` フィールドにディレクトリー名を入力します。以下に例を示します。

```

clusters:
- clusterName: "example-sno"
  networkType: "OVNKubernetes"
  extraManifestPath: extra-manifest

```

5. **SiteConfig** CR および `/extra-manifest` CR を保存し、それらをサイト設定リポジトリーにプッシュします。

GitOps ZTP パイプラインは、クラスターをプロビジョニングする際に追加のデフォルトマニフェストセットを `/extra-manifest` ディレクトリーに追加します。

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

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

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

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

また、いくつかのオプションのフィルタリングシナリオも説明されています。

前提条件

- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. GitOps ZTP パイプラインが **03-sctp-machine-config-worker.yaml** CR ファイルを適用しないようにするには、**SiteConfig** CR で次の YAML を適用します。

```

apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "site1-sno-du"
  namespace: "site1-sno-du"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.13"
  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。

```

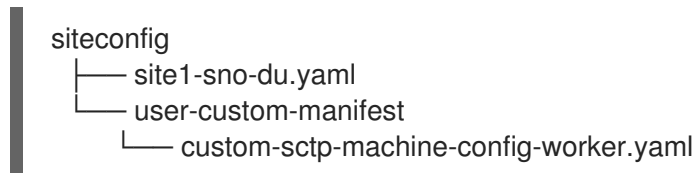
clusters:
- clusterName: "site1-sno-du"
  extraManifestPath: "<custom_manifest_folder>" ❶
  extraManifests:
    filter:
      inclusionDefault: exclude ❷
      include:
        - custom-sctp-machine-config-worker.yaml

```

❶ **<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。

❷ 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。



17.9. 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。

17.9.1. 追加の変更のクラスターへのデプロイ

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

ZTP パイプラインの完了後に追加設定を適用する

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

GitOps ZTP ライブラリーにコンテンツを追加する

GitOps ZTP パイプラインでデプロイするベースソースのカスタムリソース (CR) は、必要に応じてカスタムコンテンツで拡張できます。

クラスターインストール用の追加マニフェストの作成

インストール時に余分なマニフェストが適用され、インストール作業を効率化することができます。



重要

追加のソース CR を提供したり、既存のソース CR を変更したりすると、OpenShift Container Platform のパフォーマンスまたは CPU プロファイルに大きな影響を与える可能性があります。

関連情報

- [GitOps ZTP パイプラインでの追加インストールマニフェストのカスタマイズ](#)

17.9.2. 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:v4.13.1 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** フォルダーでは、**\$** 構文が暗示するテンプレート置換は使用されません。むしろ、**policyGen** ツールが文字列の **\$** 接頭辞を認識し、関連する **PolicyGenTemplate** CR でそのフィールドの値を指定しない場合、そのフィールドは出力 CR から完全に省かれます。

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

```

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

```

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

17.9.3. 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

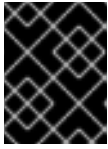
1. **PolicyGenTemplate** カスタムリソース (CR) の **kustomization.yaml** ファイルが含まれるディレクトリーに、**source-crs** という名前のサブディレクトリーを作成します。
2. 以下の例のように、カスタム CR を **source-crs** サブディレクトリーに追加します。

```

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。



重要

独自のリソースを使用するには、カスタム CR 名が ZTP コンテナで提供されるデフォルトのソース CR とは異なることを確認してください。

3. 以下の例のように、**source-crs/custom-crs** ディレクトリーに追加したコンテンツへの参照が含まれるように、必要な **PolicyGenTemplate** CR を更新します。

```

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 **fileName** を、`<subdirectory>/<filename>` など、`/source-crs` の親からのカスタム CR サブディレクトリーを含めるように設定します。

- 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 リポジトリにプッシュします。
- 次の例のように、変更された **PolicyGenTemplate** を追加して **ClusterGroupUpgrade** CR を更新し、**cgu-test.yaml** として保存します。

```

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
```

17.9.4. 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"}
```

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

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

手順

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

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

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-sno: "" 3
```

```
bindingExcludedRules:
  ztp-done: "" 4
  mcp: "master" 5
sourceFiles:
  - fileName: validatorCRs/informDuValidator.yaml
    remediationAction: inform 6
    policyName: "du-policy" 7
```

- 1 **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](#)

17.9.6. 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ではない可能性があります。

ワークロードはクリティカルまたは非クリティカルとして分類できます。クリティカルなワークロードでは、高パフォーマンスと低レイテンシーのために C ステートと P ステートの設定を無効にする必要があります。クリティカルでないワークロードでは、C ステートと P ステートの設定を使用して、いくらかのレイテンシーとパフォーマンスを犠牲にします。GitOps Zero Touch Provisioning (ZTP) を使用して、次の 3 つの電源状態を設定できます。

- 高性能モードは、最大の消費電力で超低遅延を提供します。
- パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。

- 省電力は、消費電力の削減と遅延の増加のバランスをとります。

デフォルトの設定は、低遅延のパフォーマンスモードです。

PolicyGenTemplate カスタムリソース (CR) を使用すると、**ztp-site-generate** コンテナの 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](#)

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

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

パフォーマンスモードは、比較的高い電力消費で低遅延を提供します。

前提条件

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

手順

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

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

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

17.9.6.2. 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。

17.9.6.3. 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。

- 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

- 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
```

- 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。

- /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
```

- 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

17.9.6.4. 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。

17.9.7. 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。

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-4.13
    policyName: subscription-policies
```

2. 特定のグループまたは個々のサイト設定ファイルの **spec.sourceFiles** に **LVMCluster** CR を追加します。たとえば、**group-du-sno-ranGen.yaml** ファイルに次を追加します。

```
- fileName: StorageLVMCluster.yaml
  policyName: "lvmo-config" 1
  spec:
    storage:
      deviceClasses:
        - name: vg1
          thinPoolConfig:
            name: thin-pool-1
            sizePercent: 90
            overprovisionRatio: 10
```

- 1** この設定例では、OpenShift Container Platform がインストールされているディスクを除く、使用可能なすべてのデバイスを含むボリュームグループ (**vg1**) を作成します。シンプール論理ボリュームも作成されます。

3. 必要なその他の変更およびファイルをカスタムサイトリポジトリにマージします。

4. Git で **PolicyGenTemplate** の変更をコミットし、その変更をサイト設定リポジトリにプッシュして、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。

17.9.8. 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。

17.9.8.1. 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
        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 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 **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** スタンザが存在しない場合は、**ptpClockThreshold** フィールドにデフォルト値が使用されます。スタンザは、デフォルトの **ptpClockThreshold** 値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。

2. 必要なその他の変更およびファイルをカスタムサイトリポジトリにマージします。
3. 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。

17.9.8.2. 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

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

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

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

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

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

```
- fileName: PtpConfigSlave.yaml 1
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
```

```
spec:
  profile:
    - name: "slave"
      interface: "ens5f1" 2
      ptp4IOpts: "-2 -s --summary_interval -4" 3
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 4
      ptpClockThreshold: 5
        holdOverTimeout: 30 #secs
        maxOffsetThreshold: 100 #nano secs
        minOffsetThreshold: -100 #nano secs
```

- 1 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。
 - 2 デバイス固有のインターフェイス名。
 - 3 PTP 高速イベントを有効にするには、**.spec.sourceFiles.spec.profile** の **ptp4IOpts** に **--summary_interval -4** 値を追加する必要があります。
 - 4 **phc2sysOpts** の値が必要です。**-m** はメッセージを **stdout** に出力します。**linuxptp-daemon DaemonSet** はログを解析し、Prometheus メトリックを生成します。
 - 5 オプション: **ptpClockThreshold** スタンザが存在しない場合は、**ptpClockThreshold** フィールドにデフォルト値が使用されます。スタンザは、デフォルトの **ptpClockThreshold** 値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4I**) 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。**ptp4I** 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。

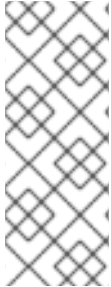
関連情報

- [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。

17.9.9. 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。

17.9.9.1. 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](#)

17.9.9.2. 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" 1
    logLevel: "info"
```

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



注記

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

4. Git で **PolicyGenTemplate** の変更をコミットし、その変更をサイト設定リポジトリにプッシュして、GitOps ZTP を使用してベアメタルイベント監視を新しいサイトにデプロイします。
5. 次のコマンドを実行して Redfish シークレットを作成します。

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password> \
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

17.9.10. イメージをローカルにキャッシュするための Image Registry Operator の設定

OpenShift Container Platform は、ローカルレジストリーを使用してイメージのキャッシュを管理します。エッジコンピューティングのユースケースでは、クラスターは集中型のイメージレジストリーと通信するときに帯域幅の制限を受けることが多く、イメージのダウンロード時間が長くなる可能性があります。

初期デプロイメント中はダウンロードに時間がかかることは避けられません。時間の経過とともに、予期しないシャットダウンが発生した場合に CRI-O が `/var/lib/containers/storage` ディレクトリーを消去するリスクがあります。イメージのダウンロード時間が長い場合の対処方法として、GitOps Zero Touch Provisioning (ZTP) を使用してリモートマネージドクラスター上にローカルイメージレジストリーを作成できます。これは、クラスターがネットワークの遠端にデプロイメントされるエッジコンピューティングシナリオで役立ちます。

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



注記

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

関連情報

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

17.9.10.1. 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 リポジトリーを作成している。

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。

17.9.10.2. 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 ①
sdb 8:16 0 446.6G 0 disk
sr0 11:0 1 104M 0 rom
```

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

関連情報

- [レジストリーへのアクセス](#)

17.9.11. 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](#)

17.9.11.1. 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}}
```

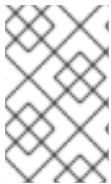
- **.ManagedClusterName** フィールドと文字列 **-name** の連結値から整数値をキャストして返します。

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

17.9.11.2. ハブクラスターテンプレートを使用したサイト PolicyGenTemplate CR でのホスト NIC の指定

単一の **ConfigMap** CR でホスト NIC を管理し、ハブクラスターテンプレートを使用して、クラスターホストに適用される生成されたポリシーにカスタム NIC 値を設定できます。サイト **PolicyGenTemplate** (PGT) CR でハブクラスターテンプレートを使用すると、サイトごとに複数の単一サイト PGT CR を作成する必要がなくなります。

次の例は、単一の **ConfigMap** CR を使用してクラスターホスト NIC を管理し、単一の **PolicyGenTemplate** サイト CR を使用してそれらをポリシーとして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

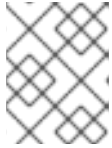
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてハブクラスターにログインしている。
- カスタムサイトの設定データを管理する Git リポジトリを作成しています。リポジトリはハブクラスターからアクセスでき、GitOps ZTP ArgoCD アプリケーションのソースリポジトリとして定義されている必要があります。

手順

1. ホストのグループの NIC を記述する **ConfigMap** リソースを作成します。以下に例を示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdata
  namespace: ztp-site
  annotations:
    argocd.argoproj.io/sync-options: Replace=true 1
data:
  example-sno-du_fh-numVfs: "8"
  example-sno-du_fh-pf: ens1f0
  example-sno-du_fh-priority: "10"
  example-sno-du_fh-vlan: "140"
  example-sno-du_mh-numVfs: "8"
  example-sno-du_mh-pf: ens3f0
  example-sno-du_mh-priority: "10"
  example-sno-du_mh-vlan: "150"
```

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



注記

ConfigMap は、ハブテンプレートの置換を持つポリシーと同じ namespace にある必要があります。

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。
3. 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サイト PGT CR を作成します。以下に例を示します。

```

apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "site"
  namespace: "ztp-site"
spec:
  remediationAction: inform
  bindingRules:
    group-du-sno: ""
    mcp: "master"
  sourceFiles:
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-fh"
      spec:
        resourceName: du_fh
        vlan: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-vlan"
.ManagedClusterName) | toInt hub}}'
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nnp-du-fh"
      spec:
        deviceType: netdevice
        isRdma: true
        nicSelector:
          pfNames:
            - '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-pf"
.ManagedClusterName) | autoindent hub}}'
          numVfs: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-numVfs"
.ManagedClusterName) | toInt hub}}'
          priority: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_fh-priority"
.ManagedClusterName) | toInt hub}}'
        resourceName: du_fh
    - fileName: SriovNetwork.yaml
      policyName: "config-policy"
      metadata:
        name: "sriov-nw-du-mh"
      spec:
        resourceName: du_mh
        vlan: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-vlan"
.ManagedClusterName) | toInt hub}}'
    - fileName: SriovNetworkNodePolicy.yaml
      policyName: "config-policy"

```

```

metadata:
  name: "sriov-nnp-du-mh"
spec:
  deviceType: vfio-pci
  isRdma: false
  nicSelector:
    pfNames:
      - '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-pf"
.ManagedClusterName) hub}}'
  numVfs: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-numVfs"
.ManagedClusterName) | toInt hub}}'
  priority: '{{hub fromConfigMap "ztp-site" "sriovdata" (printf "%s-du_mh-priority"
.ManagedClusterName) | toInt hub}}'
  resourceName: du_mh

```

4. 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。

17.9.11.3. ハブクラスターテンプレートを使用したグループ **PolicyGenTemplate** CR での VLAN ID の指定

管理対象クラスターの VLAN ID を 1 つの **ConfigMap** CR で管理し、ハブクラスターテンプレートを使用して、クラスターに適用される生成されたポリシーに VLAN ID を入力できます。

次の例は、単一の **ConfigMap** CR で VLAN ID を管理し、単一の **PolicyGenTemplate** グループ CR を使用して個々のクラスターポリシ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. クラスターホストのグループの VLAN ID を記述する **ConfigMap** CR を作成します。以下に例を示します。




```

apiVersion: v1
kind: ConfigMap
metadata:
  name: site-data
  namespace: ztp-group
  annotations:
    argocd.argoproj.io/sync-options: Replace=true ❶
data:
  site-1-vlan: "101"
  site-2-vlan: "234"

```

- 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。



注記

ConfigMap は、ハブテンプレートの置換を持つポリシーと同じ namespace にある必要があります。

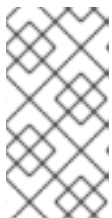
- Git で **ConfigMap** CR をコミットし、Argo CD アプリケーションによって監視されている Git リポジトリにプッシュします。
- ハブテンプレートを使用して **ConfigMap** オブジェクトから必要な VLAN ID を取得するグループ PGT CR を作成します。たとえば、次の YAML スニペットをグループ PGT CR に追加します。

```

- fileName: SriovNetwork.yaml
  policyName: "config-policy"
  metadata:
    name: "sriov-nw-du-mh"
    annotations:
      ran.openshift.io/ztp-deploy-wave: "10"
  spec:
    resourceName: du_mh
    vlan: '{{hub fromConfigMap "" "site-data" (printf "%s-vlan" .ManagedClusterName) | toInt hub}}'

```

- グループ **PolicyGenTemplate** CR を Git でコミットしてから、Argo CD 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。

17.9.11.4. 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
```

17.10. 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。

17.10.1. 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。

17.10.2. 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。

管理されたポリシーの詳細については、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。

17.10.3. 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。

17.10.4. 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.4.13.x Topology Aware Lifecycle Manager 4.13.x
Succeeded
```

2. TALM が稼働していることを確認します。

```
$ oc get deploy -n openshift-operators
```

出力例

```
NAMESPACE                                NAME                                READY UP-TO-
```

DATE AVAILABLE	AGE		
openshift-operators		cluster-group-upgrades-controller-manager	1/1
1	1	14s	

17.10.5. 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

17.10.5.1. クラスターの選択

TALM は修復計画を作成し、次のフィールドに基づいてクラスターを選択します。

- **clusterLabelSelector** フィールドは、更新するクラスターのラベルを指定します。これは、**k8s.io/apimachinery/pkg/apis/meta/v1** からの標準ラベルセレクターのリストで設定されます。リスト内の各セレクターは、ラベル値ペアまたはラベル式のいずれかを使用します。各セレクターからの一致は、**clusterSelector** フィールドおよび **cluster** フィールドからの一致と共に、クラスターの最終リストに追加されます。
- **clusters** フィールドは、更新するクラスターのリストを指定します。
- **canaries** フィールドは、カナリア更新のクラスターを指定します。
- **maxConcurrency** フィールドは、バッチで更新するクラスターの数指定します。
- **actions** フィールドは、更新プロセスを開始するときに TALM が実行する **beforeEnable** アクションと、各クラスターのポリシー修復を完了するときに TALM が実行する **afterCompletion** アクションを指定します。

clusters、**clusterLabelSelector**、および **clusterSelector** フィールドを一緒に使用して、クラスターの結合リストを作成できます。

修復計画は、**canaries** フィールドにリストされているクラスターから開始されます。各カナリアクラスターは、単一クラスターバッチを形成します。

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

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
    - ran.openshift.io/cleanup-finalizer
  generation: 1
  name: talm-cgu
  namespace: talm-namespace
  resourceVersion: '40451823'
  uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
Spec:
  actions:
    afterCompletion: 1
    addClusterLabels:
      upgrade-done: ""
    deleteClusterLabels:
      upgrade-running: ""
```

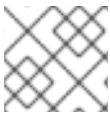
```

    deleteObjects: true
    beforeEnable: 2
    addClusterLabels:
      upgrade-running: ""
  backup: false
  clusters: 3
  - spoke1
  enable: false 4
  managedPolicies: 5
  - talm-policy
  preCaching: false
  remediationStrategy: 6
  canaries: 7
  - spoke1
  maxConcurrency: 2 8
  timeout: 240
  clusterLabelSelectors: 9
  - matchExpressions:
    - key: label1
      operator: In
      values:
        - value1a
        - value1b
  batchTimeoutAction: 10
status: 11
  computedMaxConcurrency: 2
  conditions:
  - lastTransitionTime: '2022-11-18T16:27:15Z'
    message: All selected clusters are valid
    reason: ClusterSelectionCompleted
    status: 'True'
    type: ClustersSelected 12
  - lastTransitionTime: '2022-11-18T16:27:15Z'
    message: Completed validation
    reason: ValidationCompleted
    status: 'True'
    type: Validated 13
  - lastTransitionTime: '2022-11-18T16:37:16Z'
    message: Not enabled
    reason: NotEnabled
    status: 'False'
    type: Progressing
  managedPoliciesForUpgrade:
  - name: talm-policy
    namespace: talm-namespace
  managedPoliciesNs:
  - talm-policy: talm-namespace
  remediationPlan:
  - - spoke1
    - spoke2
    - spoke3
  status:

```

1 各クラスターのポリシー修正が完了したときに TALM が実行するアクションを指定します。

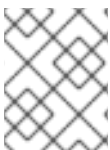
- 2 更新プロセスを開始するときに TALM が実行するアクションを指定します。
- 3 更新するクラスターの一覧を定義します。
- 4 **enable** フィールドは **false** に設定されています。
- 5 修正するユーザー定義のポリシーセットを一覧表示します。
- 6 クラスター更新の詳細を定義します。
- 7 カナリア更新のクラスターを定義します。
- 8 バッチの同時更新の最大数を定義します。修復バッチの数は、カナリアクラスターの数に加えて、カナリアクラスターを除くクラスターの数 **maxConcurrency** 値で除算します。すべての管理ポリシーに準拠しているクラスターは、修復計画から除外されます。
- 9 クラスターを選択するためのパラメーターを表示します。
- 10 バッチがタイムアウトした場合の動作を制御します。可能な値は **abort** または **continue** です。指定しない場合、デフォルトは **continue** です。
- 11 更新のステータスに関する情報を表示します。
- 12 **ClustersSelected** 条件は、選択されたすべてのクラスターが有効であることを示します。
- 13 **Validated** 条件は、選択したすべてのクラスターが検証済みであることを示します。



注記

カナリアクラスターの更新中に障害が発生すると、更新プロセスが停止します。

修復計画が正常に作成されたら、**enable** フィールドを **true** に設定できます。TALM は、指定された管理ポリシーを使用して、準拠していないクラスターの更新を開始します。



注記

ClusterGroupUpgrade CR の **enable** フィールドが **false** に設定されている場合にのみ、**spec** フィールドを変更できます。

17.10.5.2. 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。

17.10.5.3. 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。

17.10.5.4. 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。

17.10.5.5. 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 ❶
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'

```

❶ **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。

17.10.5.6. 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'

```

- 2 **Progressing** フィールドでは、更新が完了したため、ステータスは **false** です。クラスターはすべての管理ポリシーに準拠しています。
- 3 **Succeeded** フィールドは、検証が正常に完了したことを示しています。
- 1 **status** フィールドには、クラスターのリストとそれぞれのステータスが含まれます。クラスターのステータスは、**complete** または **timedout** です。

タイムアウト 状態の ClusterGroupUpgrade CR の例

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  creationTimestamp: '2022-11-18T16:27:15Z'
  finalizers:
  - ran.openshift.io/cleanup-finalizer
generation: 1
name: talm-cgu
namespace: talm-namespace
resourceVersion: '40451823'
uid: cca245a5-4bca-45fa-89c0-aa6af81a596c
spec:
  actions:
    afterCompletion:
      deleteObjects: true
    beforeEnable: {}
  backup: false
  clusters:
  - spoke1
  - spoke2
  enable: true
  managedPolicies:
  - talm-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
status:
  clusters:
  - name: spoke1
    state: complete
  - currentPolicy: 1
    name: talm-policy
    status: NonCompliant

```

```

name: spoke2
state: timedout
computedMaxConcurrency: 2
conditions:
- lastTransitionTime: '2022-11-18T16:27:15Z'
  message: All selected clusters are valid
  reason: ClusterSelectionCompleted
  status: 'True'
  type: ClustersSelected
- lastTransitionTime: '2022-11-18T16:27:15Z'
  message: Completed validation
  reason: ValidationCompleted
  status: 'True'
  type: Validated
- lastTransitionTime: '2022-11-18T16:37:16Z'
  message: Policy remediation took too long
  reason: TimedOut
  status: 'False'
  type: Progressing
- lastTransitionTime: '2022-11-18T16:37:16Z'
  message: Policy remediation took too long
  reason: TimedOut
  status: 'False'
  type: Succeeded ②
managedPoliciesForUpgrade:
- name: talm-policy
  namespace: talm-namespace
managedPoliciesNs:
  talm-policy: talm-namespace
remediationPlan:
- - spoke1
  - spoke2
status:
  startedAt: '2022-11-18T16:27:15Z'
  completedAt: '2022-11-18T20:27:15Z'

```

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

17.10.5.7. 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]' 1
    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
    reason: UpgradeNotCompleted

```

1

```

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。

17.10.6. 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-4.4.13.4
  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-4.13
            desiredUpdate:
              version: 4.4.13.4
            upstream: https://api.openshift.com/api/upgrades_info/v1/graph
        status:
          history:
            - state: Completed
              version: 4.4.13.4
          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。

17.10.6.1. 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標準になります。

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。

17.10.6.2. 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: ❶
    - 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", 1
      "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	Compliant	17m	enforce
default	cgu-policy2-common-nto-sub-policy	Compliant	17m	enforce
default	cgu-policy3-common-ntp-sub-policy	Compliant	17m	enforce
default	cgu-policy4-common-sriov-sub-policy	Compliant	17m	enforce
default	policy1-common-cluster-version-policy	NonCompliant	15h	inform
default	policy2-common-nto-sub-policy	NonCompliant	15h	inform
default	policy3-common-ntp-sub-policy	NonCompliant	18m	inform
default	policy4-common-sriov-sub-policy	NonCompliant	18m	inform

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。

- 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

- 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-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": {
  "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>
```

- b. 単一ノードクラスターに存在するすべてのサブスクリプションを確認し、以下のコマンドを実行し、**ClusterGroupUpgrade** CR でインストールしようとしているポリシーを探します。

```
$ oc get subs -A | grep -i <subscription_name>
```

cluster-logging ポリシーの出力例

NAMESPACE	NAME	PACKAGE	SOURCE
CHANNEL			
openshift-logging	cluster-logging	cluster-logging	redhat-
operators stable			

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	4.4.13.5	True	True	43s	Working towards 4.4.13.7: 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

- 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

17.10.7. 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。

17.10.7.1. 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** などのフォルダーのバックアップ。

変更されたファイルのバックアップ

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

Deployment

固定された **ostree** デプロイメント。

イメージ (オプション)

使用中のコンテナイメージ。

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。

17.10.7.2. 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 4.4.13.23 True False 86d Cluster
version is 4.4.13.23 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 4.4.13.23 True
False False 2d7h 3
clusteroperator.config.openshift.io/baremetal 4.4.13.23 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。

17.10.8. 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。

17.10.8.1. 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とするよりも多くのイメージをダウンロードします。どの事前キャッシュイメージをクラスターにダウンロードするかを制御できます。これにより、ダウンロード時間が短縮され、帯域幅とストレージが節約されます。

次のコマンドを使用して、ダウンロードするすべてのイメージのリストを表示できます。

```
$ oc adm release info <ocp-version>
```

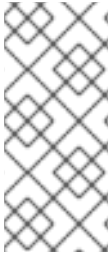
次の **ConfigMap** の例は、**excludePrecachePatterns** フィールドを使用してイメージを除外する方法を示しています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-group-upgrade-overrides
data:
  excludePrecachePatterns: |
    azure 1
    aws
    vsphere
    alibaba
```

- 1** TALM は、ここにリストされているパターンのいずれかを含む名前を持つすべてのイメージを除外します。

17.10.8.2. 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。

17.10.9. 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

17.10.9.1. 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。

17.10.9.2. 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>
```

17.10.9.3. 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

17.10.9.4. 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
```

17.10.9.5. 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
```

17.10.9.6. 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 -ojsonpath='{.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。

17.11. 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。

17.11.1. 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。

17.11.1.1. 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。
 1. 必要な OpenShift Container Platform イメージリポジトリーをミラーリングします。追加リソースにリンクされている OpenShift Container Platform イメージリポジトリーのミラーリング手順に従って、目的のプラットフォームイメージがミラーリングされていることを確認してください。**imageContentSources.yaml** ファイルの **imageContentSources** セクションの内容を保存します。

出力例

```
imageContentSources:
- mirrors:
  - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
  source: quay.io/openshift-release-dev/ocp-release
- mirrors:
  - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
  source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. ミラーリングされた目的のプラットフォーム イメージのイメージ シグネチャーを保存します。プラットフォームの更新のために、イメージ署名を **PolicyGenTemplate** CR に追加する必要があります。イメージ署名を取得するには、次の手順を実行します。

- a. 以下のコマンドを実行して、目的の OpenShift Container Platform タグを指定します。

```
$ OCP_RELEASE_NUMBER=<release_version>
```

- b. 次のコマンドを実行して、クラスターのアーキテクチャーを指定します。

```
$ ARCHITECTURE=<cluster_architecture> ❶
```

- ❶ **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-4.13 -o ~/upgrade-graph_stable-4.13
```

- 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。
- 切断されたクラスターの 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。

17.11.1.2. 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 ❶
      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-4.13"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.13
        desiredUpdate:
          version: 4.13.4
      status:
        history:
          - version: 4.13.4
            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。

- 4 更新をトリガーする **ClusterVersion** CR を示します。イメージの事前キャッシュには、**channel**、**upstream**、および **desiredVersion** フィールドがすべて必要です。

PolicyGenTemplate CR は 2 つのポリシーを生成します。

- **du-upgrade-platform-upgrade-prep** ポリシーは、プラットフォームの更新の準備作業を行います。目的のリリースイメージシグネチャーの **ConfigMap** CR を作成し、ミラー化されたリリースイメージリポジトリ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.yaml** 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.yaml
```

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。

17.11.1.3. 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:v4.13 1
        updateStrategy: 2
        registryPoll:
          interval: 1h

```

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

2 Operator Lifecycle Manager (OLM) が新しい Operator バージョンのインデックスイメージをポーリングする頻度を **registryPoll.interval** フィールドで設定します。y-stream および z-stream Operator の更新のために新しいインデックスイメージタグが常にプッシュされる場合、この変更は必要ありません。**registryPoll.interval** フィールドを短い間隔に設定して更新を促進できますが、間隔を短くすると計算負荷が増加します。これに対処するために、更新が完了したら、**registryPoll.interval** をデフォルト値に戻すことができます。

- 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 4.13 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 ①
  - common-subscriptions-policy ②
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

- ① このポリシーは、カタログソースから Operator イメージを取得するために、イメージの事前キャッシュ機能で必要になります。

- 2 ポリシーには Operator サブスクリプションが含まれます。参照 **PolicyGenTemplates** の構造と内容に従っている場合、すべての Operator サブスクリプション



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](#)。

17.11.1.3.1. 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atalogソース設定の名前を入力します。

17.11.1.4. 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。

別添条件

- 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。

17.11.1.5. 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。

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。

17.11.2. 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

```

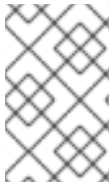
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" ❶
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" ❷
  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

```

- ❶ 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。
- ❷ 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。

17.12. 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。

17.12.1. 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。

17.12.2. 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。

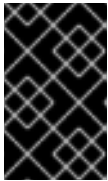
```
$ mkdir -p ./update
```

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.13 extract /home/ztp --tar | tar x -C ./update
```

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

- **update/extra-manifest: SiteConfig** CR が追加のマニフェスト **configMap** を生成するために使用するソース CR ファイルが含まれています。

- **update/source-crs** には、**PolicyGenTemplate** CR が Red Hat Advanced Cluster Management (RHACM) ポリシーを生成するために使用するソース CR ファイルが含まれています。
 - **update/argocd/deployment** には、この手順の次のステップで使用するハブクラスターに適用するパッチおよび YAML ファイルが含まれます。
 - **update/argocd/example**: 推奨される設定を表す **SiteConfig** および **PolicyGenTemplate** ファイルの例が含まれています。
3. **clusters-app.yaml** ファイルおよび **policies-app.yaml** ファイルを更新して、Git リポジトリーのアプリケーションおよび URL、ブランチ、およびパスを反映します。
アップグレードにポリシーの廃止につながる変更が含まれている場合は、アップグレードを実行する前に、廃止されたポリシーを削除する必要があります。
 4. **/update** フォルダー内の設定およびデプロイソース CR と、フリーサイト CR を管理する Git リポジトリーとの間の変更を比較します。必要な変更をサイトリポジトリーに適用してプッシュします。

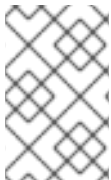


重要

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

17.12.3. 既存クラスターのラベル付け

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



注記

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

手順

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

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

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

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

17.12.4. 既存の GitOps ZTP アプリケーションの停止

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

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

手順

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

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

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

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

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

17.12.5. Git リポジトリーに必要な変更

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 リポジトリーのコンテンツに関する追加の要件があります。これらの変更を反映するには、リポジトリー内の既存のコンテンツを更新する必要があります。

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

```

├── 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。

17.12.6. 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. **update/argocd/deployment/** ディレクトリーに以前に展開したパッチファイルを使用して、ハブクラスターの ArgoCD インスタンスにパッチを適用するには、以下のコマンドを入力します。

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

2. **argocd/deployment** ディレクトリーの内容を適用するには、以下のコマンドを入力します。

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

17.12.7. 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。

17.13. 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。

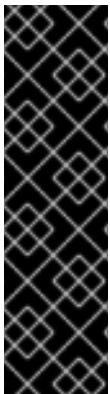


注記

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の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、サ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については、[シングルノードの 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。

17.13.1. 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。アップグレードプロセスを開始するには、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。

17.13.2. (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

DU プロファイルが GitOps Zero Touch Provisioning (ZTP) プラグインバージョン 4.11 以前を使用してデプロイされた場合、PTP および SR-IOV Operator は、**master** というラベルの付いたノードにのみデーモンを配置するように設定されている可能性があります。この設定により、PTP および SR-IOV デーモンがワーカーノードで動作しなくなります。システムで PTP および SR-IOV デーモンノードセ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 ptpoperatorconfig/default -n openshift-ptp -ojsonpath='{.spec}' | jq
```

PTP Operator の出力例

```
{"daemonNodeSelector":{"node-role.kubernetes.io/master":""}} 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。

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。

17.13.3. 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)。

17.13.4. 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:
  name: "example-sno-workers"
  namespace: "example-sno"
spec:
  bindingRules:
    sites: "example-sno" ①
  mcp: "worker" ②
  sourceFiles:
    - fileName: MachineConfigGeneric.yaml ③
      policyName: "config-policy"
      metadata:
        labels:
          machineconfiguration.openshift.io/role: worker
        name: enable-workload-partitioning
      spec:

```

```

config:
  storage:
    files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmlldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWVudC
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbylKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzliA9IDAsICJjcHVzZXQiID0gljAtMyIgfQo=
          mode: 420
          overwrite: true
          path: /etc/crio/crio.conf.d/01-workload-partitioning
          user:
            name: root
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6IiwLTMiCiAgfQp9Cg==
          mode: 420
          overwrite: true
          path: /etc/kubernetes/openshift-workload-pinning
          user:
            name: root
    - fileName: PerformanceProfile.yaml
      policyName: "config-policy"
      metadata:
        name: openshift-worker-node-performance-profile
      spec:
        cpu: 4
          isolated: "4-47"
          reserved: "0-3"
        hugepages:
          defaultHugepagesSize: 1G
        pages:
          - size: 1G
            count: 32
        realTimeKernel:
          enabled: true
    - fileName: TunedPerformancePatch.yaml
      policyName: "config-policy"
      metadata:
        name: performance-patch-worker
      spec:
        profile:
          - name: performance-patch-worker
            data: |
              [main]
              summary=Configuration changes profile inherited from performance created tuned
              include=openshift-node-performance-openshift-worker-node-performance-profile
              [bootloader]
              cmdline_crash=nohz_full=4-47 5
              [sysctl]
              kernel.timer_migration=1
              [scheduler]
              group.ice-ptp=0:f:10*:ice-ptp.*
              [service]
              service.stalld=start,enable

```

```
service.chrondy=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
```

17.13.5. GitOps ZTP を使用して単一ノードの OpenShift クラスターにワーカーノードを追加する

1つ以上のワーカーノードを既存の単一ノード OpenShift クラスターに追加して、クラスターで使用可能な CPU リソースを増やすことができます。

前提条件

- OpenShift Container Platform 4.11 以降のベアメタルハブクラスターに RHACM 2.6 以降をインストールして設定する
- ハブクラスターに Topology Aware Lifecycle Manager をインストールする
- ハブクラスターに Red Hat OpenShift GitOps をインストールする
- GitOps ZTP **ztp-site-generate** コンテナ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		69m	
example-node2	provisioning	true		4m50s	1

- 1 provisioning ステータスは、インストールメディアからのノードの起動が進行中であることを示します。

- インストールプロセスを継続的に監視します。
 - 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

- 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":""}
```

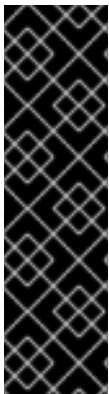
17.14. 単一ノードの OPENSIFT デプロイメント用のイメージの事前キャッシュ

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。

17.14.1. factory-precaching-cli ツールの入手

factory-precaching-cli ツールの Go バイナリーは、[Telco RAN 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
```

17.14.2. ライブオペレーティングシステムイメージからの起動

factory-precaching-cli ツールを使用して、1つのディスクしか使用できず、外部ディスクドライブをサーバーに接続できないサーバーを起動できます。



警告

RHCOS では、ディスクが RHCOS イメージで書き込まれようとしているときに、ディスクが使用されていない必要があります。

サーバーハードウェアに応じて、次のいずれかの方法を使用して、空のサーバーに RHCOS ライブ ISO をマウントできます。

- Dell サーバーで Dell RACADM ツールを使用する。
- HP サーバーで HPONCFG ツールを使用する。
- Redfish BMC API を使用する。



注記

マウント手順を自動化することを推奨します。手順を自動化するには、必要なイメージをプルして、ローカル HTTP サーバーでホストする必要があります。

前提条件

- ホストの電源を入れた。
- ホストへのネットワーク接続がある。



手順

この例の手順では、Redfish BMC API を使用して RHCOS ライブ ISO をマウントします。

1. RHCOS ライブ ISO をマウントします。
 - a. 仮想メディアのステータスを確認します。

```
$ curl --globoff -H "Content-Type: application/json" -H \
  "Accept: application/json" -k -X GET --user ${username_password} \
  https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1 | python -m json.tool
```

- b. ISO ファイルを仮想メディアとしてマウントします。

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\\n" -ku ${username_password} -H
"Content-Type: application/json" -H "Accept: application/json" -d '{"Image":
"http://[$HTTPd_IP]/RHCOS-live.iso"}' -X POST
https://$BMC_ADDRESS/redfish/v1/Managers/Self/VirtualMedia/1/Actions/VirtualMedia.Ins
ertMedia
```

- c. 仮想メディアから1回起動するように起動順序を設定します。

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\\n" -ku ${username_password} -H
"Content-Type: application/json" -H "Accept: application/json" -d '{"Boot":{
"BootSourceOverrideEnabled": "Once", "BootSourceOverrideTarget": "Cd",
"BootSourceOverrideMode": "UEFI"}}' -X PATCH
https://$BMC_ADDRESS/redfish/v1/Systems/Self
```

2. 再起動し、サーバーが仮想メディアから起動していることを確認します。

関連情報

- **butane** ユーティリティの詳細については、[Butane について](#) を参照してください。
- カスタムライブ RHCOS ISO の作成の詳細については、[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 の使用の詳細については、[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。

17.14.3. 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

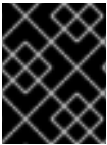
```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
loop0 7:0 0 93.8G 0 loop /run/ephemeral
loop1 7:1 0 897.3M 1 loop /sysroot
sr0 11:0 1 999M 0 rom /run/media/iso
nvme0n1 259:1 0 1.5T 0 disk
```

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。

17.14.3.1. 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
```

17.14.3.2. 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。

17.14.4. 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

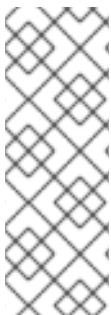


注記

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。

17.14.4.1. 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
```

17.14.4.2. 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。

17.14.4.3. 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 4.13.0 \ 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:                4.13.0
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 ❶
```

- ❶ /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

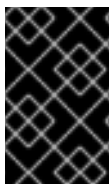
-rw-r--r--. 1 root root 145535253 Oct 31 15:26 ocp-v4.0-art-
dev@sha256_f8f098911d670287826e9499806553f7a1dd3e2b5332abbec740008c36e84de5.t
gz
-rw-r--r--. 1 root root 158048761 Oct 31 15:40 ocp-v4.0-art-
dev@sha256_f914228d8bb99120986262168a705903a9f49724ffa958bb4bf12b2ec1d7fb47.tgz
```

```
-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
```

17.14.4.4. 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 \ 1
-r 4.13.0 \ 2
--acm-version 2.6.3 \ 3
--mce-version 2.1.4 \ 4
-f /mnt \ 5
--img quay.io/custom/repository 6
--du-profile -s 7
```

- 1 factory-precaching-cli ツールのダウンロード機能を指定します。
- 2 OpenShift Container Platform リリースバージョンを定義します。
- 3 RHACM バージョンを定義します。
- 4 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。

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:                4.13.0
Hub Version:            2.6.3
ACM Version:            2.6.3
MCE Version:            2.1.4
Include DU Profile:     Yes
Workers:                83

```

17.14.4.5. 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 \ ❶
-r 4.13.0 \ 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
--generate-imageset ❽

```

- 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-4.13
        minVersion: 4.13.0 1
        maxVersion: 4.13.0
  additionalImages:
    - name: quay.io/custom/repository
  operators:
    - catalog: registry.redhat.io/redhat/redhat-operator-index:v4.13
      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-4.13'
- 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:v4.13
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せん。
- 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:v4.13
  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 4.13.0 \ 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 4.13.0 \
--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。

17.14.5. 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=xf\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\nTimeoutStopSec=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-
ai-
images.txt%7D%22%0ABINARY_FOLDER%3D%2Fvar%2Fmnt%0A%0Apushd%20%24FOLDER%0
A%0Atotal_copies%3D%24%28sort%20-
u%20%24BINARY_FOLDER%2F%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%20zxf%20
%24%7Btar%7D.tgz%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20%5B
%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20copy%20
dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-
storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-
eq%200%20%5D%3B%20then%20rm%20-
rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29
%3B%20fi%0A%20done%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCP_RELEASE_LI
ST%7D%0A%0A%23%20workaround%20while%20https%3A%2F%2Fgithub.com%2Fopenshift%2Ffa
ssisted-service%2Fpull%2F3546%0A%23cp%20%2Fvar%2Fmnt%2Fmodified-rhcos-4.10.3-x86_64-
metal.x86_64.raw.gz%20%2Fvar%2Ftmp%2F.%0A%0Aexit%200}}},
{"overwrite":true,"path":"/usr/local/bin/agent-fix-bz1964591","mode":755,"user":
{"name":"root"},"contents":
{"source":"data:,%23%21%2Fusr%2Fbin%2Fsh%0A%0A%23%20This%20script%20is%20a%20work
around%20for%20bugzilla%201964591%20where%20symlinks%20inside%20%2Fvar%2Flib%2Fcont
ainers%2F%20get%0A%23%20corrupted%20under%20some%20circumstances.%0A%23%0A%23%
20In%20order%20to%20let%20agent.service%20start%20correctly%20we%20are%20checking%20h
ere%20whether%20the%20requested%0A%23%20container%20image%20exists%20and%20in%20c
ase%20%22podman%20images%22%20returns%20an%20error%20we%20try%20removing%20the
%20faulty%0A%23%20image.%0A%23%0A%23%20In%20such%20a%20scenario%20agent.service
%20will%20detect%20the%20image%20is%20not%20present%20and%20pull%20it%20again.%20In
%20case%0A%23%20the%20image%20is%20present%20and%20can%20be%20detected%20correc
tly%2C%20no%20any%20action%20is%20required.%0A%0AIMAGE%3D%24%28echo%20%241%2
0%7C%20sed%20%27s%2F%3A.%2A%2F%2F%27%29%0A%20podman%20image%20exists%20%24I
MAGE%20%7C%7C%20echo%20%22already%20loaded%22%20%7C%7C%20echo%20%22need
%20to%20be%20pulled%22%0A%23podman%20images%20%7C%20grep%20%24IMAGE%20%7C
%7C%20podman%20rmi%20--force%20%241%20%7C%7C%20true}}}}'
nodes:
- hostname: "snode.sno-worker-0.example.domain.redhat.com"
  role: "master"
  bmcAddress: "idrac-virtualmedia+https://10.19.28.53/redfish/v1/Systems/System.Embedded.1"
  bmcCredentialsName:
    name: "worker0-bmh-secret"
  bootMACAddress: "e4:43:4b:bd:90:46"
  bootMode: "UEFI"
  rootDeviceHints:
    deviceName: /dev/nvme0n1
  cpuset: "0-1,40-41"
  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=xf\nTimeoutSec=30\n\n[Install]\nRequiredBy=precache-ocp-

```



```

images.service"}, {"name": "precache-ocp-images.service", "enabled": true, "contents": "[Unit]
Description=Extracts the precached OCP images into containers storage
After=var-mnt.mount
Before=machine-config-daemon-pull.service nodeip-configuration.service
[Service]
Type=oneshot
User=root
WorkingDirectory=/var/mnt
ExecStart=ash /usr/local/bin/extract-ocp.sh
TimeoutStopSec=60
[Install]
WantedBy=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%0A%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%20total%0Acurrent_copy%3D1%0A%0Awhile%20read%20-r%20line%3B%0A%0A%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%7C%20awk%20%27%7Bprint%242%7D%27%29%0A%20%20podman%20image%20exists%20%24uri%0A%20%20if%20%5B%5B%20%24%3F%20-eq%200%20%5D%5D%3B%20then%0A%20%20%20%20%20echo%20%22Skipping%20existing%20image%20%24tar%22%0A%20%20%20%20%20echo%20%22Copying%20%24%7Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20%20%20%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%0A%20%20%20%20%20continue%0A%20%20fi%0A%20%20tar%3D%24%28echo%20%22%24uri%22%20%7C%200%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%20%24%3F%20-eq%200%20%5D%3B%20then%20rm%20-f%20%24%7Btar%7D.gz%3B%20fi%0A%20%20echo%20%22Copying%20%24%7Buri%7D%20%5B%24%7Bcurrent_copy%7D%2F%24%7Btotal_copies%7D%5D%22%0A%20%20skopeo%20copy%20dir%3A%2F%2F%24%28pwd%29%2F%24%7Btar%7D%20containers-storage%3A%24%7Buri%7D%0A%20%20if%20%5B%20%24%3F%20-eq%200%20%5D%3B%20then%20rm%20-rf%20%24%7Btar%7D%3B%20current_copy%3D%24%28%28current_copy%20%2B%201%29%29%3B%20fi%0Adone%20%3C%20%24%7BBINARY_FOLDER%7D%2F%24%7BOCF_RELEASE_LIST%7D%0A%0Aexit%200"}]]}'
nodeNetwork:
  config:
    interfaces:
      - name: ens1f0
        type: ethernet
        state: up
        macAddress: "AA:BB:CC:11:22:33"
        ipv4:
          enabled: true
          dhcp: true
        ipv6:
          enabled: false
    interfaces:
      - name: "ens1f0"
        macAddress: "AA:BB:CC:11:22:33"

```

- 1 **spec.clusters.clusterImageSetNameRef** フィールドに別のイメージセットを指定しない限り、デプロイメントに使用されるクラスターイメージセットを指定します。
- 2 個々のクラスターをデプロイするために使用されるクラスターイメージセットを指定します。定義されている場合には、サイトレベルで **spec.clusters.clusterImageSetNameRef** を上書きします。

されている場合には、サイトレベルで `spec.clusterimagesetnamekey` を上書きします。

17.14.5.1. 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。

17.14.5.2. 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

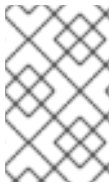
...

--insecure-ignition

Allow Ignition URL without HTTPS or hash

17.14.5.3. nodes.ignitionConfigOverride フィールドについて

clusters.ignitionConfigOverride と同様に、**nodes.ignitionConfigOverride** フィールドを使用すると、Ignition 形式の設定を **coreos-installer** ユーティリティーに追加できます。ただし、これを追加できるのは、OpenShift Container Platform のインストール段階です。RHCOS がディスクに書き込まれると、GitOps ZTP 検出 ISO に含まれる追加の設定は使用できなくなります。検出段階で、追加の設定はライブ OS のメモリーに保存されます。



注記

この段階では、展開およびロードされたコンテナイメージの数は、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。

17.14.6. トラブルシューティング

17.14.6.1. 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 4.13.0 --acm-version 2.5.4 \
--mce-version 2.0.4 -f /mnt --img quay.io/custom/repository
--du-profile -s --skip-imageset
```