OpenShift Container Platform 4.10

スケーラビリティー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

Last Updated: 2023-11-07

OpenShift Container Platform 4.10 スケーラビリティー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 © 2023 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 ホストについての推奨されるプラクティス **5**
1.1. ノードホストについての推奨プラクティス 5
1.2. KUBELET パラメーターを編集するための KUBELETCONFIG CRD の作成 6
1.3. 利用不可のワーカーノードの数の変更 10
1.4. コントロールプレーンノードのサイジング 11
1.5. ETCD についての推奨されるプラクティス 14
1.6. ETCD を別のディスクに移動する 16
1.7. ETCD データのデフラグ 19
1.8. OPENSHIFT CONTAINER PLATFORM インフラストラクチャーコンポーネント 23
1.9. モニタリングソリューションの移動 24
1.10. デフォルトレジストリーの移行 26
1.11. ルーターの移動 28
1.12. インフラストラクチャーノードのサイジング 30
1.13. 関連情報 31

第2章 IBM Z および LINUXONE 環境に推奨されるホストプラクティス **32**
2.1. CPU のオーバーコミットの管理 32
2.2. TRANSPARENT HUGE PAGES (THP) の無効 33
2.3. RECEIVE FLOW STEERING を使用したネットワークパフォーマンスの強化 33
2.4. ネットワーク設定の選択 34
2.5. 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。 35
2.6. IBM Z ホストの RHEL KVM の推奨事項 36

第3章 クラスタースケーリングに関する推奨プラクティス **41**
3.1. クラスターのスケーリングに関する推奨プラクティス 41
3.2. マシンセットの変更 41
3.3. マシンのヘルスチェック 43
3.4. サンプル MACHINEHEALTHCHECK リソース 44
3.5. MACHINEHEALTHCHECK リソースの作成 46

第4章 NODE TUNING OPERATOR の使用 **47**
4.1. NODE TUNING OPERATOR について 47
4.2. NODE TUNING OPERATOR 仕様サンプルへのアクセス 47
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 48
4.4. TUNED プロファイルが適用されていることの確認 48
4.5. カスタムチューニング仕様 49
4.6. カスタムチューニングの例 53
4.7. サポートされている TUNED デーモンプラグイン 54

第5章 CPU マネージャーおよび TOPOLOGY MANAGER の使用 **56**
5.1. CPU マネージャーの設定 56
5.2. TOPOLOGY MANAGER ポリシー 60
5.3. TOPOLOGY MANAGER のセットアップ 61
5.4. POD の TOPOLOGY MANAGER ポリシーとの対話 61

第6章 NUMA 対応ワークロードのスケジューリング **63**
6.1. NUMA 対応のスケジューリングについて 63
6.2. NUMA RESOURCES OPERATOR のインストール 64
6.3. NUMARESOURCESOPERATOR カスタムリソースの作成 67
6.4. NUMA 対応のセカンダリー POD スケジューラーのデプロイ 68
6.5. 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 70
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 73

第7章 CLUSTER MONITORING OPERATOR のスケーリング **83**
7.1. PROMETHEUS データベースのストレージ要件 83
7.2. クラスターモニタリングの設定 84

第8章 オブジェクトの最大値に合わせた環境計画 **86**
8.1. メジャーリリースについての OPENSHIFT CONTAINER PLATFORM のテスト済みクラスターの最大値 86
8.2. クラスターの最大値がテスト済みの OPENSHIFT CONTAINER PLATFORM 環境および設定 88
8.3. テスト済みのクラスターの最大値に基づく環境計画 90
8.4. 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 91

第9章 ストレージの最適化 **94**
9.1. 利用可能な永続ストレージオプション 94
9.2. 設定可能な推奨のストレージ技術 95
9.3. データストレージ管理 97
9.4. MICROSOFT AZURE のストレージパフォーマンスの最適化 98

第10章 ルーティングの最適化 **100**
10.1. 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 100

第11章 ネットワークの最適化 **102**
11.1. ネットワークでの MTU の最適化 102
11.2. 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 103
11.3. IPSEC の影響 103

第12章 ベアメタルホストの管理 **104**
12.1. ベアメタルホストおよびノードについて 104
12.2. ベアメタルホストのメンテナンス 104

第13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 **109**
13.1. HUGE PAGE の機能 109
13.2. HUGE PAGE がアプリケーションによって消費される仕組み 109
13.3. DOWNWARD API を使用した HUGE PAGE リソースの使用 110
13.4. HUGE PAGE の設定 112
13.5. TRANSPARENT HUGE PAGES (THP) の無効化 114

第14章 低レイテンシーのノード向けの PERFORMANCE ADDON OPERATOR **116**
14.1. 低レイテンシー 116
14.2. PERFORMANCE ADDON OPERATOR のインストール 117
14.3. PERFORMANCE ADDON OPERATOR のアップグレード 119
14.4. 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 122
14.5. 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 135
14.6. PERFORMANCE ADDON OPERATOR を使用した NIC キューの削減 140
14.7. 低レイテンシー CNF チューニングステータスのデバッグ 147
14.8. 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 149

第15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 **152**
15.1. レイテンシーテストを実行するための前提条件 152
15.2. レイテンシーテストの検出モードについて 152
15.3. レイテンシーの測定 153
15.4. レイテンシーテストの実行 154
15.5. レイテンシーテストの失敗レポートの生成 167
15.6. JUNIT レイテンシーテストレポートの生成 168
15.7. 単一ノードの OPENSHIFT クラスターでレイテンシーテストを実行する 168
15.8. 切断されたクラスターでのレイテンシーテストの実行 169
15.9. CNF-TESTS コンテナーでのエラーのトラブルシューティング 172

第16章 クラスター更新のための TOPOLOGY AWARE LIFECYCLE MANAGER 173
16.1. TOPOLOGY AWARE LIFECYCLE MANAGER の設定について 173
16.2. TOPOLOGY AWARE LIFECYCLE MANAGER で使用される管理ポリシー 173
16.3. WEB コンソールを使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール 174
16.4. CLI を使用した TOPOLOGY AWARE LIFECYCLE MANAGER のインストール 175
16.5. CLUSTERGROUPUPGRADE CR 176
16.6. マネージドクラスターでのポリシーの更新 188
16.7. コンテナーイメージ事前キャッシュ機能の使用 195
16.8. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング 198

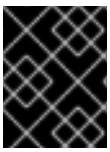
第17章 パフォーマンスプロファイルの作成 207
17.1. PERFORMANCE PROFILE CREATOR の概要 207
17.2. 関連情報 218

第18章 単一ノード OPENSHIFT でのワークロードパーティション設定 219
18.1. ワークロードの分割による CPU 割り当ての最大化 219

第19章 ネットワーク遠端のクラスター 222
19.1. ネットワークファー遠端の課題 222
19.2. ZTP 用のハブクラスターの準備 227
19.3. 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 234
19.4. 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 245
19.5. 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 257
19.6. 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 OPENSHIFT クラスター設定 268
19.7. VDU アプリケーションワークロードの単一ノード OPENSHIFT クラスターチューニングの検証 290
19.8. 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 305
19.9. 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 307
19.10. TOPOLOGY AWARE LIFECYCLE MANAGER を使用したマネージドクラスターの更新 316
19.11. GITOPS ZTP の更新 332

第1章 ホスト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。



重要

これらのガイドラインは、Open Virtual Network (OVN) ではなく、ソフトウェア定義ネットワーク (SDN) を使用する OpenShift Container Platform に該当します。

1.1. ノードホストについての推奨プラクティス

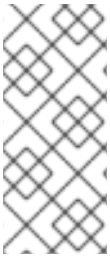
OpenShift Container Platform ノードの設定ファイルには、重要なオプションが含まれています。たとえば、**podsPerCore** および **maxPods** の 2 つのパラメーターはノードにスケジュールできる Pod の最大数を制御します。

両方のオプションが使用されている場合、2 つの値の低い方の値により、ノード上の Pod 数が制限されます。これらの値を超えると、以下の状態が生じる可能性があります。

- CPU 使用率の増大。
- Pod のスケジューリングの速度が遅くなる。
- (ノードのメモリー量によって) メモリー不足のシナリオが生じる可能性。
- IP アドレスのプールを消費する。
- リソースのオーバーコミット、およびこれによるアプリケーションのパフォーマンスの低下。

重要

Kubernetes では、単一コンテナーを保持する Pod は実際には 2 つのコンテナーを使用します。2 つ目のコンテナーは実際のコンテナーの起動前にネットワークを設定するために使用されます。そのため、10 の Pod を使用するシステムでは、実際には 20 のコンテナーが実行されていることになります。



注記

クラウドプロバイダーからのディスク IOPS スロットリングは CRI-O および kubelet に影響を与える可能性があります。ノード上に多数の I/O 集約型 Pod が実行されている場合、それらはオーバーロードする可能性があります。ノード上のディスク I/O を監視し、ワークロード用に十分なスループットを持つボリュームを使用することが推奨されます。

podsPerCore は、ノードのプロセッサーコア数に基づいてノードが実行できる Pod 数を設定します。たとえば、4 プロセッサーコアを搭載したノードで **podsPerCore** が **10** に設定される場合、このノードで許可される Pod の最大数は **40** になります。

```
kubeletConfig:
  podsPerCore: 10
```

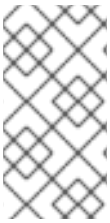
podsPerCore を **0** に設定すると、この制限が無効になります。デフォルトは **0** です。**podsPerCore** は **maxPods** を超えることができません。

maxPods は、ノードのプロパティーにかかわらず、ノードが実行できる Pod 数を固定値に設定します。

```
kubeletConfig:
  maxPods: 250
```

1.2. KUBELET パラメーターを編集するための KUBELETCONFIG CRD の作成

kubelet 設定は、現時点で Ignition 設定としてシリアル化されているため、直接編集することができます。ただし、新規の **kubelet-config-controller** も Machine Config Controller (MCC) に追加されます。これにより、**KubeletConfig** カスタムリソース (CR) を使用して kubelet パラメーターを編集できます。



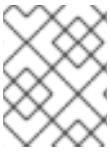
注記

kubeletConfig オブジェクトのフィールドはアップストリーム Kubernetes から kubelet に直接渡されるため、kubelet はそれらの値を直接検証します。**kubeletConfig** オブジェクトに無効な値により、クラスターノードが利用できなくなります。有効な値は、Kubernetes ドキュメント を参照してください。

以下のガイダンスを参照してください。

- マシン設定プールごとに、そのプールに加える設定変更をすべて含めて、**KubeletConfig** CR を 1 つ作成します。同じコンテンツをすべてのプールに適用している場合には、すべてのプールに **KubeletConfig** CR を 1 つだけ設定する必要があります。
- 既存の **KubeletConfig** CR を編集して既存の設定を編集するか、変更ごとに新規 CR を作成する代わりに新規の設定を追加する必要があります。CR を作成するのは、別のマシン設定プールを変更する場合、または一時的な変更を目的とした変更の場合のみにして、変更を元に戻すことができるようにすることを推奨します。
- 必要に応じて、クラスターごとに 10 を制限し、複数の **KubeletConfig** CR を作成します。最初の **KubeletConfig** CR について、Machine Config Operator (MCO) は **kubelet** で追加されたマシン設定を作成します。それぞれの後続の CR で、コントローラーは数字の接尾辞が付いた別の **kubelet** マシン設定を作成します。たとえば、**kubelet** マシン設定があり、その接尾辞が **-2** の場合に、次の **kubelet** マシン設定には **-3** が付けられます。

マシン設定を削除する場合は、制限を超えないようにそれらを逆の順序で削除する必要があります。たとえば、**kubelet-3** マシン設定を、**kubelet-2** マシン設定を削除する前に削除する必要があります。



注記

接尾辞が **kubelet-9** のマシン設定があり、別の **KubeletConfig** CR を作成する場合には、**kubelet** マシン設定が 10 未満の場合でも新規マシン設定は作成されません。

KubeletConfig CR の例

```
$ oc get kubeletconfig
```

```
NAME             AGE
set-max-pods     15m
```

KubeletConfig マシン設定を示す例

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1                 b5c5119de007945b6fe6fb215db3b8e2ceb12511    3.2.0
26m
...
```

以下の手順は、ワーカーノードでノードあたりの Pod の最大数を設定する方法を示しています。

前提条件

1. 設定するノードタイプの静的な **MachineConfigPool** CR に関連付けられたラベルを取得します。以下のいずれかの手順を実行します。

   a. マシン設定プールを表示します。

   ```
   $ oc describe machineconfigpool <name>
   ```

   以下に例を示します。

   ```
   $ oc describe machineconfigpool worker
   ```

   出力例

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: 2019-02-08T14:52:39Z
     generation: 1
     labels:
       custom-kubelet: set-max-pods 1
   ```

   1 ラベルが追加されると、**labels** の下に表示されます。

   b. ラベルが存在しない場合は、キー/値のペアを追加します。

   ```
   $ oc label machineconfigpool worker custom-kubelet=set-max-pods
   ```

手順

1. これは、選択可能なマシン設定オブジェクトを表示します。

   ```
   $ oc get machineconfig
   ```

   デフォルトで、2 つの kubelet 関連の設定である **01-master-kubelet** および **01-worker-kubelet** を選択できます。

2. ノードあたりの最大 Pod の現在の値を確認します。

```
$ oc describe node <node_name>
```

以下に例を示します。

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

Allocatable スタンザで **value: pods: <value>** を検索します。

出力例

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                         3500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      15341844Ki
 pods:                        250
```

3. ワーカーノードでノードあたりの最大の Pod を設定するには、kubelet 設定を含むカスタムリソースファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods 1
  kubeletConfig:
    maxPods: 500 2
```

1. Machine Config Pool からラベルを入力します。
2. kubelet 設定を追加します。この例では、**maxPods** を使用してノードあたりの最大 Pod を設定します。

注記

kubelet が API サーバーと通信する速度は、1秒あたりのクエリー (QPS) およびバースト値により異なります。デフォルト値の **50** (**kubeAPIQPS** の場合) および **100** (**kubeAPIBurst** の場合) は、各ノードで制限された Pod が実行されている場合には十分な値です。ノード上に CPU およびメモリーリソースが十分にある場合には、kubelet QPS およびバーストレートを更新することが推奨されます。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. ラベルを使用してワーカーのマシン設定プールを更新します。

```
$ oc label machineconfigpool worker custom-kubelet=large-pods
```

b. **KubeletConfig** オブジェクトを作成します。

```
$ oc create -f change-maxPods-cr.yaml
```

c. **KubeletConfig** オブジェクトが作成されていることを確認します。

```
$ oc get kubeletconfig
```

出力例

```
NAME              AGE
set-max-pods      15m
```

クラスター内のワーカーノードの数によっては、ワーカーノードが 1 つずつ再起動されるのを待機します。3 つのワーカーノードを持つクラスターの場合は、10 分 から 15 分程度かかる可能性があります。

4. 変更がノードに適用されていることを確認します。

a. **maxPods** 値が変更されたワーカーノードで確認します。

```
$ oc describe node <node_name>
```

b. **Allocatable** スタンザを見つけます。

```
 ...
Allocatable:
  attachable-volumes-gce-pd:  127
```

```
cpu:                    3500m
ephemeral-storage:      123201474766
hugepages-1Gi:          0
hugepages-2Mi:          0
memory:                 14225400Ki
pods:                   500 (1)
...
```

(1) この例では、**pods** パラメーターは **KubeletConfig** オブジェクトに設定した値を報告するはずです。

5. **KubeletConfig** オブジェクトの変更を確認します。

```
$ oc get kubeletconfigs set-max-pods -o yaml
```

これは、以下の例のように **True** および **type:Success** のステータスを表示します。

```
spec:
  kubeletConfig:
    maxPods: 500
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
status:
  conditions:
  - lastTransitionTime: "2021-06-30T17:04:07Z"
    message: Success
    status: "True"
    type: Success
```

1.3. 利用不可のワーカーノードの数の変更

デフォルトでは、kubelet 関連の設定を利用可能なワーカーノードに適用する場合に 1 つのマシンのみを利用不可の状態にすることが許可されます。大規模なクラスターの場合、設定の変更が反映されるまでに長い時間がかかる可能性があります。プロセスのスピードを上げるためにマシン数の調整をいつでも実行することができます。

手順

1. **worker** マシン設定プールを編集します。

```
$ oc edit machineconfigpool worker
```

2. **maxUnavailable** を必要な値に設定します。

```
spec:
  maxUnavailable: <node_count>
```

重要

値を設定する際に、クラスターで実行されているアプリケーションに影響を与えずに利用不可にできるワーカーノードの数を検討してください。

1.4. 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	64
501	4000	16	96

3 つのコントロールプレーンノード (またはマスターノード) 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、コストを節約するためにシャットダウンした後にクラスターが再起動する意図的なケースに加えて、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が発生すること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、マスターが遮断 (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。

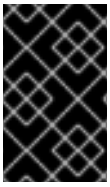
namespace 数	アイドル状態の OLM メモリー (GB)	ユーザー Operator が 5 つインストールされている OLM メモリー (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6

重要

以下の設定でのみ、実行中の OpenShift Container Platform 4.10 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。
- インストーラーによってプロビジョニングされたインフラストラクチャーインストール方法でインストールされた AWS 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 4.10 では、デフォルトで CPU コア (500 ミリコア) の半分がシステムによって予約されます (OpenShift Container Platform 3.11 以前のバージョンと比較)。サイズはこれを考慮に入れて決定されます。

1.4.1. 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。

1.4.1.1. 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

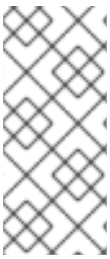
1.5. ETCD についての推奨されるプラクティス

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ーは、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 20 ミリ秒の場合、fdatasync 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。

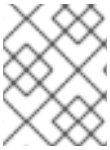
次のハードディスク機能は、最適な etcd パフォーマンスを提供します。

- 高速読み取り操作をサポートするための低レイテンシー。
- 圧縮と最適化を高速化するための高帯域幅書き込み。
- 障害からの回復を高速化するための高帯域幅読み取り。
- 最低限の選択肢としてソリッドステートドライブがありますが、NVMe ドライブが推奨されます。
- 信頼性を高めるためのさまざまなメーカーのサーバーグレードのハードウェア。
- パフォーマンス向上のための RAID0 テクノロジー。
- 専用の etcd ドライブ。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。

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 NVM 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ーを使用して、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ーを使用すると、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。

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 などのコンテナーランタイムは、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/openshift-scale/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/openshift-scale/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 パーセンタイルの比較でディスクが 20 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** メトリクスは、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

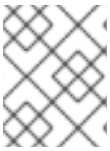
- **fio** を使用して OpenShiftContainerPlatform で OpenShift Container Platform ディスクのパフォーマンスを確認する方法

1.6. 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。

前提条件

- **MachineConfigPool** は **metadata.labelsmachineconfiguration.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。
- **/dev/sdb** などのノードの補助記憶装置は、sdb と一致する必要があります。ファイル内のすべての場所でこの参照を変更します。



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。

Machine Config Operator (MCO) は、OpenShift Container Platform 4.10 コンテナーストレージのセカンダリーディスクのマウントを担当します。

次の手順を使用して、etcd を別のデバイスに移動します。

手順

1. **etcd-mc.yml** という名前の **machineconfig** YAML ファイルを作成して、次の情報を追加します。

```
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/sdb
          DefaultDependencies=no
          BindsTo=dev-sdb.device
          After=dev-sdb.device var.mount
          Before=systemd-fsck@dev-sdb.service

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

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

          [Mount]
          What=/dev/sdb
          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=/usr/sbin/setenforce 0
    ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
    ExecStart=/usr/sbin/setenforce 1
    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
```

2. 次のコマンドを入力して、マシン設定を作成します。

```
$ oc login -u ${ADMIN} -p ${ADMINPASSWORD} ${API}
... output omitted ...
```

```
$ oc create -f etcd-mc.yml
machineconfig.machineconfiguration.openshift.io/98-var-lib-etcd created
```

```
$ oc login -u ${ADMIN} -p ${ADMINPASSWORD} ${API}
 [... output omitted ...]
```

```
$ oc create -f etcd-mc.yml machineconfig.machineconfiguration.openshift.io/98-var-lib-etcd
created
```

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。

3. ノードが別のディスクに配置されたら、マシン設定ファイル **etcd-mc.yml** を次の情報で更新します。

```
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/sdb to /var/lib/etcd
          Before=local-fs.target
          Requires=systemd-mkfs@dev-sdb.service
          After=systemd-mkfs@dev-sdb.service var.mount

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

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

4. 次のコマンドを入力して、デバイスを作成および同期するためのロジックを削除する変更されたバージョンを適用します。

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

前の手順により、ノードが再起動されなくなります。

関連情報

- Red Hat Enterprise Linux CoreOS (RHCOS)

1.7. 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。

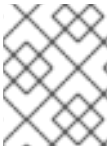
これらの主要な指標をモニターします。

- **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。

1.7.1. 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 コントローラー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>**

1.7.2. 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 |   3.4.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  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 コンテナーに接続し、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 |   3.4.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |   41 MB |     false |      false |
7 |      91624 |              91624 |        | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  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
```

1.8. OPENSHIFT CONTAINER PLATFORM インフラストラクチャーコンポーネント

以下のインフラストラクチャーワークロードでは、OpenShift Container Platform ワーカーのサブスクリプションは不要です。

- マスターで実行される Kubernetes および OpenShift Container Platform コントロールプレーンサービス
- デフォルトルーター
- 統合コンテナーイメージレジストリー
- HAProxy ベースの Ingress Controller
- ユーザー定義プロジェクトのモニタリング用のコンポーネントを含む、クラスターメトリクスの収集またはモニタリングサービス
- クラスター集計ロギング
- サービスブローカー
- Red Hat Quay
- Red Hat OpenShift Data Foundation
- Red Hat Advanced Cluster Manager
- Kubernetes 用 Red Hat Advanced Cluster Security
- Red Hat OpenShift GitOps
- Red Hat OpenShift Pipelines

他のコンテナー、Pod またはコンポーネントを実行するノードは、サブスクリプションが適用される必要のあるワーカーノードです。

インフラストラクチャーノードおよびインフラストラクチャーノードで実行できるコンポーネントの詳細は、OpenShift sizing and subscription guide for enterprise Kubernetes の"Red Hat OpenShift control plane and infrastructure nodes"セクションを参照してください。

1.9. モニタリングソリューションの移動

監視スタックには、Prometheus、Grafana、Alertmanager などの複数のコンポーネントが含まれています。Cluster Monitoring Operator は、このスタックを管理します。モニタリングスタックをインフラストラクチャーノードに再デプロイするために、カスタム config map を作成して適用できます。

手順

1. **cluster-monitoring-config** 設定マップを編集し、**nodeSelector** を変更して **infra** ラベルを使用します。

```
$ oc edit configmap cluster-monitoring-config -n openshift-monitoring
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
```

```
  nodeSelector: (1)
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
prometheusK8s:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
prometheusOperator:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
grafana:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
k8sPrometheusAdapter:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoExecute
kubeStateMetrics:
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - key: node-role.kubernetes.io/infra
    value: reserved
    effect: NoSchedule
```

```
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  telemeterClient:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  openshiftStateMetrics:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
  thanosQuerier:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoExecute
```

1 1 適切な値が設定された **nodeSelector** パラメーターを、移動する必要のあるコンポーネントに追加します。表示されている形式の **nodeSelector** を使用することも、ノードに指定された値に基づいて **<key>: <value>** ペアを使用することもできます。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

2. モニタリング Pod が新規マシンに移行することを確認します。

```
$ watch 'oc get pod -n openshift-monitoring -o wide'
```

3. コンポーネントが **infra** ノードに移動していない場合は、このコンポーネントを持つ Pod を削除します。

```
$ oc delete pod -n openshift-monitoring <pod>
```

削除された Pod からのコンポーネントが **infra** ノードに再作成されます。

1.10. デフォルトレジストリーの移行

レジストリー Operator を、その Pod を複数の異なるノードにデプロイするように設定します。

前提条件

- 追加のマシンセットを OpenShift Container Platform クラスターに設定します。

手順

1. **config/instance** オブジェクトを表示します。

```
$ oc get configs.imageregistry.operator.openshift.io/cluster -o yaml
```

出力例

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12ffeee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
...
```

2. **config/instance** オブジェクトを編集します。

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
          topologyKey: kubernetes.io/hostname
        weight: 100
  logLevel: Normal
  managementState: Managed
```

```
nodeSelector: 1
  node-role.kubernetes.io/infra: ""
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/infra
  value: reserved
- effect: NoExecute
  key: node-role.kubernetes.io/infra
  value: reserved
```

1 適切な値が設定された **nodeSelector** パラメーターを、移動する必要のあるコンポーネントに追加します。表示されている形式の **nodeSelector** を使用することも、ノードに指定された値に基づいて **<key>: <value>** ペアを使用することもできます。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

3. レジストリー Pod がインフラストラクチャーノードに移動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、レジストリー Pod が置かれているノードを特定します。

   ```
   $ oc get pods -o wide -n openshift-image-registry
   ```

   b. ノードに指定したラベルがあることを確認します。

   ```
   $ oc describe node <node_name>
   ```

   コマンド出力を確認し、**node-role.kubernetes.io/infra** が **LABELS** リストにあることを確認します。

1.11. ルーターの移動

ルーター Pod を異なるマシンセットにデプロイできます。デフォルトで、この Pod はワーカーノードにデプロイされます。

前提条件

- 追加のマシンセットを OpenShift Container Platform クラスターに設定します。

手順

1. ルーター Operator の **IngressController** カスタムリソースを表示します。

   ```
   $ oc get ingresscontroller default -n openshift-ingress-operator -o yaml
   ```

   コマンド出力は以下のテキストのようになります。

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     creationTimestamp: 2019-04-18T12:35:39Z
     finalizers:
     - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
     generation: 1
     name: default
   ```

```
  namespace: openshift-ingress-operator
  resourceVersion: "11341"
  selfLink: /apis/operator.openshift.io/v1/namespaces/openshift-ingress-
operator/ingresscontrollers/default
  uid: 79509e05-61d6-11e9-bc55-02ce4781844a
spec: {}
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2019-04-18T12:36:15Z
    status: "True"
    type: Available
  domain: apps.<cluster>.example.com
  endpointPublishingStrategy:
    type: LoadBalancerService
  selector: ingresscontroller.operator.openshift.io/deployment-ingresscontroller=default
```

2. **ingresscontroller** リソースを編集し、 **nodeSelector** を **infra** ラベルを使用するように変更します。

```
$ oc edit ingresscontroller default -n openshift-ingress-operator
```

```
spec:
  nodePlacement:
    nodeSelector: (1)
      matchLabels:
        node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

(1) 適切な値が設定された **nodeSelector** パラメーターを、移動する必要のあるコンポーネントに追加します。表示されている形式の **nodeSelector** を使用することも、ノードに指定された値に基づいて **<key>: <value>** ペアを使用することもできます。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

3. ルーター Pod が **infra** ノードで実行されていることを確認します。

   a. ルーター Pod のリストを表示し、実行中の Pod のノード名をメモします。

```
$ oc get pod -n openshift-ingress -o wide
```

出力例

```
NAME                              READY   STATUS        RESTARTS   AGE     IP          NODE
NOMINATED NODE   READINESS GATES
router-default-86798b4b5d-bdlvd   1/1     Running       0          28s     10.130.2.4  ip-10-
0-217-226.ec2.internal   <none>           <none>
router-default-955d875f4-255g8    0/1     Terminating   0          19h     10.129.2.4  ip-10-
0-148-172.ec2.internal   <none>           <none>
```

この例では、実行中の Pod は **ip-10-0-217-226.ec2.internal** ノードにあります。

b. 実行中の Pod のノードのステータスを表示します。

```
$ oc get node <node_name> 1
```

1 Pod のリストより取得した **<node_name>** を指定します。

出力例

```
NAME                           STATUS  ROLES        AGE   VERSION
ip-10-0-217-226.ec2.internal   Ready   infra,worker  17h   v1.23.0
```

ロールのリストに **infra** が含まれているため、Pod は正しいノードで実行されます。

1.12. 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 のメトリック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。以下のインフラストラクチャーノードのサイズの推奨内容は、クラスターの最大値およびコントロールプレーンの密度に重点を置いたテストの結果に基づいています。

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 4.10 では、デフォルトで CPU コア (500 ミリコア) の半分がシステムによって予約されます (OpenShift Container Platform 3.11 以前のバージョンと比較)。これは、上記のサイジングの推奨内容に影響します。

1.13. 関連情報

- 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

第2章 IBM Z および LINUXONE 環境に推奨されるホストプラクティス

このトピックでは、IBM Z および 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 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。

2.1. 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。論理プロセッサーが LPAR に定義されているよりも多くの仮想プロセッサー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ー数を設定し、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

2.2. 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。

2.3. 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。

2.3.1. 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-rps.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ー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

2.4. 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

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

2.5. 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。

2.5.1. 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トノードの YAML 設定を追加する必要があります。

手順

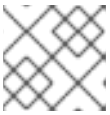
1. 以下の MCO サンプルプロファイルをコントロールプレーンノードの YAML ファイルにコピーします。たとえば、**05-master-kernelarg-hpav.yaml** です。

```
$ 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トノードの 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

2.6. 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 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。

2.6.1. VirtIO ネットワークインターフェイスに複数のキューを使用

複数の仮想 CPU を使用すると、受信パケットおよび送信パケットに複数のキューを指定すると、パッケージを並行して転送できます。**driver** 要素の **queues** 属性を使用して複数のキューを設定します。仮想サーバーの仮想 CPU の数を超えない 2 以上の整数を指定します。

以下の仕様の例では、ネットワークインターフェイスの入出力キューを 2 つ設定します。

```
<interface type="direct">
   <source network="net01"/>
   <model type="virtio"/>
```

```
    <driver ... queues="2"/>
</interface>
```

複数のキューは、ネットワークインターフェイス用に強化されたパフォーマンスを提供するように設計されていますが、メモリーおよび CPU リソースも使用します。ビジーなインターフェイス用の 2 つのキューの定義を開始します。次に、トラフィックが少ないインターフェイスの場合は 2 つのキューを、ビジーなインターフェイスの場合は 3 つ以上のキューを試してください。

2.6.2. 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>
  <iothreads>3</iothreads> (1)
    ...
    <devices>
      ...
        <disk type="block" device="disk"> (2)
<driver ... iothread="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 iothreadadd 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。

2.6.3. 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。ホスト上でバッキング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。

2.6.4. 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" iothread="1"/>
...
</disk>
```

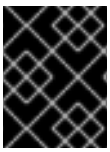
2.6.5. メモリーバルーンデバイスを除外します。

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"/>
```

2.6.6. 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。

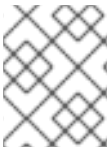
間隔を 60000 ns に動的に設定するには、以下のコマンドを入力します。

```
# 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
```

2.6.7. 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。

2.6.8. 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3章 クラスタースケーリングに関する推奨プラクティス

重要

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。

これらのガイドラインは、Open Virtual Network (OVN) ではなく、ソフトウェア定義ネットワーク (SDN) を使用する OpenShift Container Platform に該当します。

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。

3.1. クラスターのスケーリングに関する推奨プラクティス

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。

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。

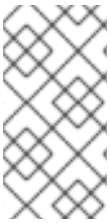
注記

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。デフォルトのクライアント QPS およびバーストレートは、現時点で **5** と **10** にそれぞれ設定されています。これらは OpenShift Container Platform で変更することはできません。

3.2. マシンセットの変更

マシンセットを変更するには、**MachineSet** YAML を編集します。次に、各マシンを削除するか、マシンセットを **0** レプリカにスケールダウンしてマシンセットに関連付けられたすべてのマシンを削除します。レプリカは必要な数にスケーリングします。マシンセットへの変更は既存のマシンに影響を与えません。

他の変更を加えずに、マシンセットをスケーリングする必要がある場合、マシンを削除する必要はありません。



注記

デフォルトで、OpenShift Container Platform ルーター Pod はワーカーにデプロイされます。ルーターは Web コンソールなどの一部のクラスターリソースにアクセスすることが必要であるため、 ルーター Pod をまず再配置しない限り、ワーカーのマシンセットを **0** にスケーリングできません。

前提条件

- OpenShift Container Platform クラスターおよび **oc** コマンドラインをインストールすること。
- **cluster-admin** パーミッションを持つユーザーとして、**oc** にログインする。

手順

1. マシンセットを編集します。

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

2. マシンセットを **0** にスケールダウンします。

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

または、以下を実行します。

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

マシンが削除されるまで待機します。

3. マシンセットを随時スケールアップします。

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

または、以下を実行します。

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

マシンが起動するまで待ちます。新規マシンにはマシンセットに加えられた変更が含まれます。

3.3. マシンのヘルスチェック

マシンのヘルスチェックは特定のマシンプールの正常ではないマシンを自動的に修復します。

マシンの正常性を監視するには、リソースを作成し、コントローラーの設定を定義します。5 分間 **NotReady** ステータスにすることや、 node-problem-detector に永続的な条件を表示すること、および監視する一連のマシンのラベルなど、チェックする条件を設定します。

注記

マスターロールのあるマシンにマシンヘルスチェックを適用することはできません。

MachineHealthCheck リソースを監視するコントローラーは定義済みのステータスをチェックします。マシンがヘルスチェックに失敗した場合、このマシンは自動的に検出され、その代わりとなるマシンが作成されます。マシンが削除されると、**machine deleted** イベントが表示されます。

マシンの削除による破壊的な影響を制限するために、コントローラーは 1 度に 1 つのノードのみをドレイン (解放) し、これを削除します。マシンのターゲットプールで許可される **maxUnhealthy** しきい値を上回る数の正常でないマシンがある場合、修復が停止するため、手動による介入が可能になります。

注記

タイムアウトについて注意深い検討が必要であり、ワークロードと要件を考慮してください。

- タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。
- タイムアウトが短すぎると、修復ループが生じる可能性があります。たとえば、**NotReady** ステータスを確認するためのタイムアウトについては、マシンが起動プロセスを完了できるように十分な時間を設定する必要があります。

チェックを停止するには、リソースを削除します。

3.3.1. マシンヘルスチェックのデプロイ時の制限

マシンヘルスチェックをデプロイする前に考慮すべき制限事項があります。

- マシンセットが所有するマシンのみがマシンヘルスチェックによって修復されます。
- コントロールプレーンマシンは現在サポートされておらず、それらが正常でない場合にも修正されません。
- マシンのノードがクラスターから削除される場合、マシンヘルスチェックはマシンが正常ではないとみなし、すぐにこれを修復します。
- **nodeStartupTimeout** の後にマシンの対応するノードがクラスターに加わらない場合、マシンは修復されます。
- **Machine** リソースフェーズが **Failed** の場合、マシンはすぐに修復されます。

3.4. サンプル MACHINEHEALTHCHECK リソース

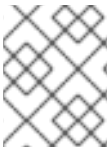
ベアメタルを除くすべてのクラウドベースのインストールタイプの **MachineHealthCheck** リソースは、以下の YAML ファイルのようになります。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role> 3
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" 5
    status: "False"
  - type:    "Ready"
    timeout: "300s" 6
    status: "Unknown"
  maxUnhealthy: "40%" 7
  nodeStartupTimeout: "10m" 8
```

1. デプロイするマシンヘルスチェックの名前を指定します。
2. 3. チェックする必要のあるマシンプールのラベルを指定します。
4. 追跡するマシンセットを **<cluster_name>-<label>-<zone>** 形式で指定します。たとえば、**prod-node-us-east-1a** とします。
5. 6. ノードの状態のタイムアウト期間を指定します。タイムアウト期間の条件が満たされると、マシンは修正されます。タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。

7 ターゲットプールで同時に修復できるマシンの数を指定します。これはパーセンテージまたは整数として設定できます。正常でないマシンの数が **maxUnhealthy** で設定された制限を超える場合、

8 マシンが正常でないと判別される前に、ノードがクラスターに参加するまでマシンヘルスチェックが待機する必要のあるタイムアウト期間を指定します。



注記

matchLabels はあくまでもサンプルであるため、特定のニーズに応じてマシングループをマッピングする必要があります。

3.4.1. マシンヘルスチェックによる修復の一時停止 (short-circuiting)

一時停止 (short-circuiting) が実行されることにより、マシンのヘルスチェックはクラスターが正常な場合にのみマシンを修復するようになります。一時停止 (short-circuiting) は、**MachineHealthCheck** リソースの **maxUnhealthy** フィールドで設定されます。

ユーザーがマシンの修復前に **maxUnhealthy** フィールドの値を定義する場合、**MachineHealthCheck** は **maxUnhealthy** の値を、正常でないと判別するターゲットプール内のマシン数と比較します。正常でないマシンの数が **maxUnhealthy** の制限を超える場合、修復は実行されません。

重要

maxUnhealthy が設定されていない場合、値は **100%** にデフォルト設定され、マシンはクラスターの状態に関係なく修復されます。

適切な **maxUnhealthy** 値は、デプロイするクラスターの規模や、**MachineHealthCheck** が対応するマシンの数によって異なります。たとえば、**maxUnhealthy** 値を使用して複数のアベイラビリティーゾーン間で複数のマシンセットに対応でき、ゾーン全体が失われると、**maxUnhealthy** の設定によりクラスター内で追加の修復を防ぐことができます。複数のアベイラビリティーゾーンを持たないグローバル Azure リージョンでは、アベイラビリティーセットを使用して高可用性を確保できます。

maxUnhealthy フィールドは整数またはパーセンテージのいずれかに設定できます。**maxUnhealthy** の値によって、修復の実装が異なります。

3.4.1.1. 絶対値を使用した maxUnhealthy の設定

maxUnhealthy が **2** に設定される場合:

- 2 つ以下のノードが正常でない場合に、修復が実行されます。
- 3 つ以上のノードが正常でない場合は、修復は実行されません。

これらの値は、マシンヘルスチェックによってチェックされるマシン数と別個の値です。

3.4.1.2. パーセンテージを使用した maxUnhealthy の設定

maxUnhealthy が **40%** に設定され、25 のマシンがチェックされる場合:

- 10 以下のノードが正常でない場合に、修復が実行されます。
- 11 以上のノードが正常でない場合は、修復は実行されません。

maxUnhealthy が **40%** に設定され、6 マシンがチェックされる場合:

- 2 つ以下のノードが正常でない場合に、修復が実行されます。
- 3 つ以上のノードが正常でない場合は、修復は実行されません。

注記

チェックされる **maxUnhealthy** マシンの割合が整数ではない場合、マシンの許可される数は切り捨てられます。

3.5. MACHINEHEALTHCHECK リソースの作成

クラスターに、すべての **MachineSets** の **MachineHealthCheck** リソースを作成できます。コントロールプレーンマシンをターゲットとする **MachineHealthCheck** リソースを作成することはできません。

前提条件

- **oc** コマンドラインインターフェイスをインストールします。

手順

1. マシンヘルスチェックの定義を含む **healthcheck.yml** ファイルを作成します。
2. **healthcheck.yml** ファイルをクラスターに適用します。

```
$ oc apply -f healthcheck.yml
```

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 デーモンのオーケストレーションによるノードレベルのチューニングの管理に役立ちます。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、コンテナー化された 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ー化された 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。

コンテナー化された 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ー化された 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、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。

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

1. 以下を実行します。

```
$ oc get Tuned/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されます。

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 の今後のバージョンで非推奨になる場合があります。

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:
  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 -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。

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、コンテナー化された 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)
```

(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。

<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

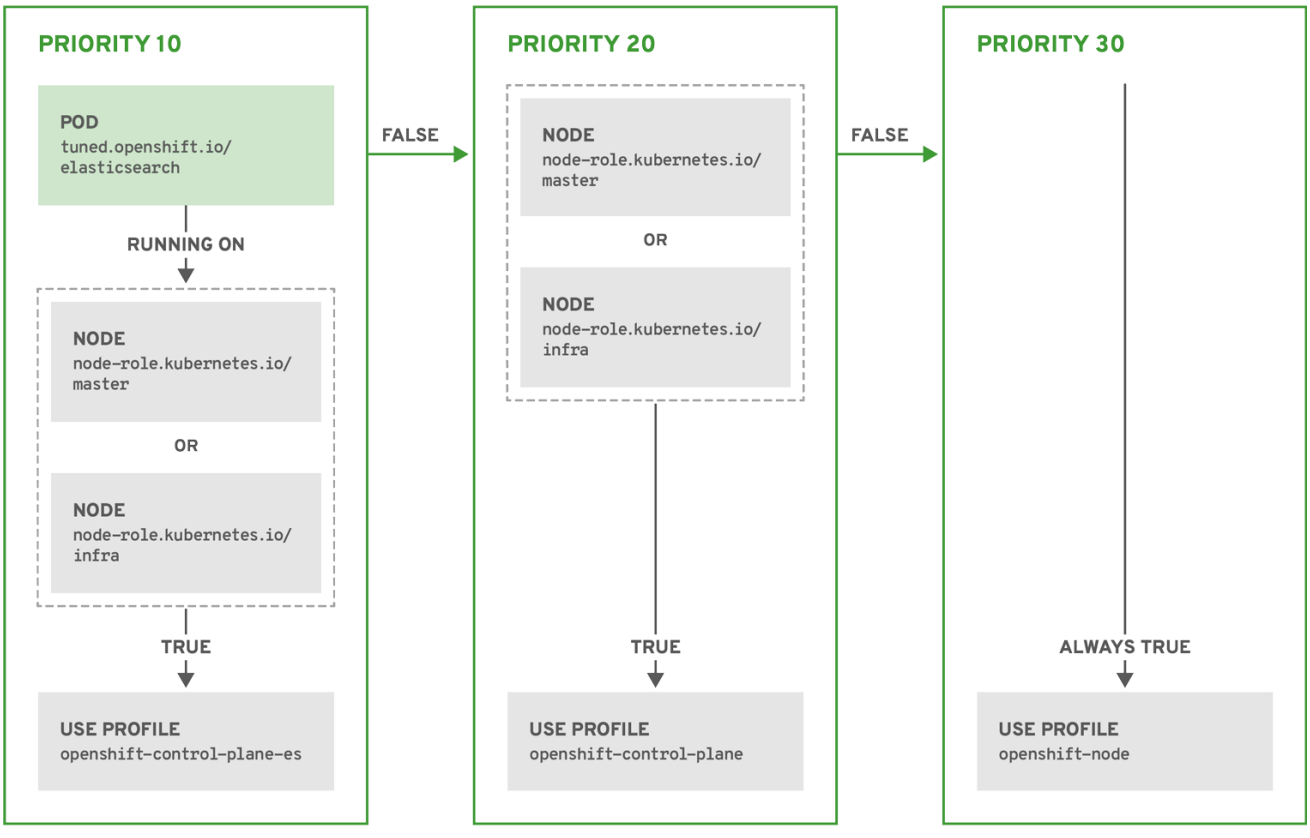
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra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

```
    profile: openshift-control-plane
  - priority: 30
    profile: openshift-node
```

上記のコンテナー化された 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ー化された 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、コンテナー化された 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。



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レ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。

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、コンピュートノードに OpenShift 固有のチューニングを使用します。

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。

その他の参考資料

- 利用可能な TuneD プラグイン
- TuneD を使い始める

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
- プロセッサー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、単一のプロセッサー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 1
     cpuManagerReconcilePeriod: 5s 2
```

1 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ーを提供しません。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 内のコンテナーを許可します。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。

2 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 つまたは複数を要求する 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ル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 マネージャーやデバイスマネージャーなどの 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ーの場合、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、そのコンテナーの推奨される NUMA ノードのアフィニティーを保存します。アフィニティー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 を許可します。

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ーの場合、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、そのコンテナーの推奨される NUMA ノードのアフィニティーを保存します。アフィニティー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ーの場合、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ー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ー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、作成できます。

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

Topololgy 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す。

```
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。

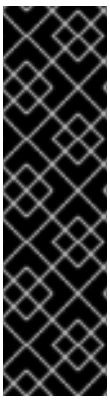
```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

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デバイスマネージャー) 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、このコンテナー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 クラスター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。



重要

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

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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。

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ノードの 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

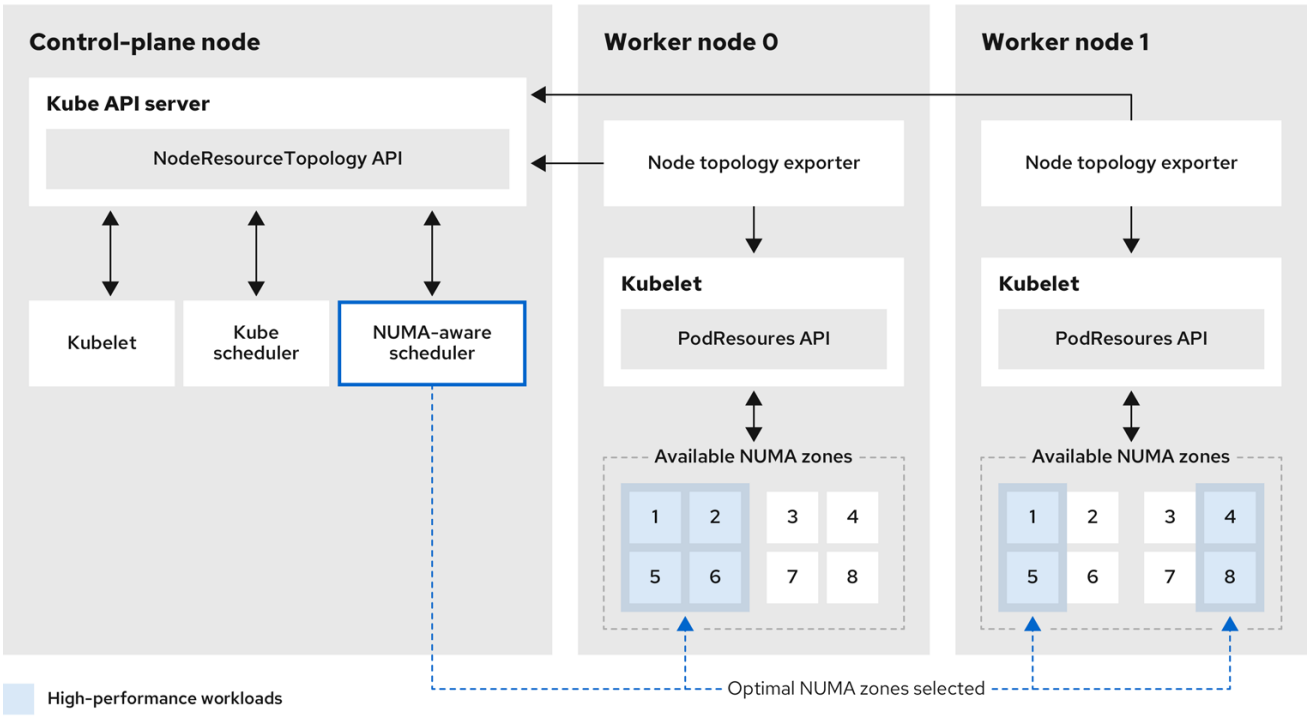
Non-Uniform Memory Access (NUMA) は、異なる CPU が異なるメモリー領域に異なる速度でアクセスできるようにするコンピュートプラットフォームアーキテクチャーです。NUMA リソーストポロジーは、コンピュートノード内の相互に関連する CPU、メモリー、および PCI デバイスの位置を指しています。共同配置されたリソースは、同じ **NUMA ゾーン** にあるとされています。高性能アプリケーションの場合、クラスターは単一の NUMA ゾーンで Pod ワークロードを処理する必要があります。

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ー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。

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、クラスター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ー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



NodeResourceTopology API

NodeResourceTopology API は、各コンピュートノードで使用可能な 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ルします。

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、各コンピュートノードで使用可能な 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。

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: "{product-version}"
  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.v4.10.0  NUMA Resources Operator      4.10.0              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. OpenShift Container Platform Web コンソールを使用して 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。

2. 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 **InstallSucceeded** の **Status** で **default** 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、**default** プロジェクトの Pod のログを確認します。

6.3. 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. 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。
   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/v1alpha1
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. 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。

- 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" 1
    cpuManagerReconcilePeriod: "5s"
    reservedSystemCPUs: "0,1"
    memoryManagerPolicy: "Static" 2
    evictionHard:
      memory.available: "100Mi"
    kubeReserved:
      memory: "512Mi"
    reservedMemory:
      - numaNode: 0
        limits:
          memory: "1124Mi"
    systemReserved:
      memory: "512Mi"
    topologyManagerPolicy: "single-numa-node" 3
    topologyManagerScope: "pod"
```

1. **cpuManagerPolicy** の場合、**static** は小文字の **s** を使用する必要があります。
2. **memoryManagerPolicy** の場合、**Static** は大文字の **S** を使用する必要があります。
3. **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/v1alpha1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.10"
```

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
```

6.5. 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: ctnr
        image: quay.io/openshifttest/hello-openshift:openshift
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            memory: "100Mi"
            cpu: "10"
          requests:
            memory: "100Mi"
            cpu: "10"
      - name: ctnr2
        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 をスケジュール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リスト表示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 describe noderesourcetopologies.topology.node.k8s.io
```

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、次のコマンドを実行して、Pod に **Guaranteed** の **qosClass** があることを確認します。

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

出力例

```
Guaranteed
```

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

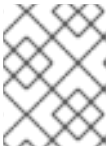
```
topo-aware-scheduler
```

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。

4. 次のコマンドを実行して、スケジュール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/v1alpha1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoawareschedwg/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/v1alpha1
  kind: NodeResourceTopology
  metadata:
    annotations:
```

```
    k8stopoawareschedwg/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** 以下の各スタンザは、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/v1alpha1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.10"
  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。

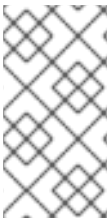
```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

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タインスタンスには、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** コンテナーのログを調べます。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

```
I0221 13:38:18.334140       1 main.go:206] using sysinfo:
reservedCpus: 0,1
reservedMemory:
  "0": 1178599424
I0221 13:38:18.334370       1 main.go:67] === System information ===
I0221 13:38:18.334381       1 sysinfo.go:231] cpus: reserved "0-1"
I0221 13:38:18.334493       1 sysinfo.go:237] cpus: online "0-103"
I0221 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ター設定マップの修正

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ター (RTE) デーモンセット 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 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、システムリソース、コンテナー、およびコンポーネントのメトリックを 1 つのダッシュボードインターフェイスである Grafana で表示できます。

7.1. PROMETHEUS データベースのストレージ要件

Red Hat では、異なるスケールサイズに応じて各種のテストが実行されました。

注記

以下の Prometheus ストレージ要件は規定されていません。ワークロードのアクティビティーおよびリソースの使用に応じて、クラスターで観察されるリソースの消費量が大きくなる可能性があります。

表7.1 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	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	RAM 領域 (スケールサイズに基づく)	ネットワーク (tsdb チャンクに基づく)
50	1800	6.3 GB	94 GB	6 GB	16 MB
100	3600	13 GB	195 GB	10 GB	26 MB
150	5400	19 GB	283 GB	12 GB	36 MB
200	7200	25 GB	375 GB	14 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。
- NVMe (non-volatile memory express) ドライブを搭載した 3 つ以上の **openshift-container-storage** ノードを使用します。

7.2. 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}} 1
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: {{STORAGE_CLASS}} 2
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} 3
    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. 標準の値は **PROMETHEUS_RETENTION_PERIOD=15d** になります。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
```

第8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。

重要

これらのガイドラインは、Open Virtual Network (OVN) ではなく、ソフトウェア定義ネットワーク (SDN) を使用する OpenShift Container Platform に該当します。

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ことがあります。

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

OpenShift Container Platform 3.x のテスト済みクラウドプラットフォーム: Red Hat OpenStack (RHOSP)、Amazon Web Services および Microsoft AzureOpenShift Container Platform 4.x のテスト済み Cloud Platform : Amazon Web Services、 Microsoft Azure および Google Cloud Platform

最大値のタイプ	3.x テスト済みの最大値	4.x テスト済みの最大値
ノード数	2,000	2,000 [1]
Pod の数[2]	150,000	150,000
ノードあたりの Pod 数	250	500 [3]
コアあたりの Pod 数	デフォルト値はありません。	デフォルト値はありません。
namespace の数[4]	10,000	10,000
ビルド数	10,000(デフォルト Pod RAM 512 Mi)- Pipeline ストラテジー	10,000(デフォルト Pod RAM 512 Mi)- Source-to-Image (S2I) ビルドストラテジー

最大値のタイプ	3.x テスト済みの最大値	4.x テスト済みの最大値
namespace ごとの Pod の数[5]	25,000	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	ルーターあたり 2,000
シークレットの数	80,000	80,000
config map の数	90,000	90,000
サービスの数[6]	10,000	10,000
namespace ごとのサービス数	5,000	5,000
サービスごとのバックエンド数	5,000	5,000
namespace ごとのデプロイメントの数[5]	2,000	2,000
ビルド設定の数	12,000	12,000
カスタムリソース定義 (CRD) の数	デフォルト値はありません。	512 [7]

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。

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ー、CPU、ストレージ要件により異なります。

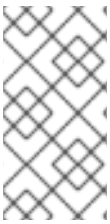
3. これは、ワーカーノードごとに 500 の Pod を持つ 100 ワーカーノードを含むクラスターでテストされています。デフォルトの **maxPods** は 250 です。500 **maxPods** に到達するには、クラスターはカスタム kubelet 設定を使用し、**maxPods** が **500** に設定された状態で作成される必要があります。500 ユーザー Pod が必要な場合は、ノード上に 10-15 のシステム Pod がすでに実行されているため、**hostPrefix** が **22** である必要があります。永続ボリューム要求 (PVC) が割り当てられている Pod の最大数は、PVC の割り当て元のストレージバックエンドによって異なります。このテストでは、OpenShift Data Foundation v4 (OCS v4) のみが、本書で説明されているノードごとの Pod 数に対応することができました。

4. 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。

5. システムには、状態の変更に対する対応として特定の namespace にある全オブジェクトに対して反復する多数のコントロールループがあります。単一の namespace に特定タイプのオブジェクトの数が多くなると、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。

6. 各サービスポートと各サービスのバックエンドには、iptables の対応するエントリーがあります。特定のサービスのバックエンド数は、エンドポイントのオブジェクトサイズに影響があり、その結果、システム全体に送信されるデータサイズにも影響を与えます。

7. OpenShift Container Platform には、OpenShift Container Platform によってインストールされたもの、OpenShift Container Platform と統合された製品、およびユーザー作成の CRD を含め、合計 512 のカスタムリソース定義 (CRD) の制限があります。512 を超える CRD が作成されている場合は、**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。



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るためです。

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

8.2.1. 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)/IOS	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
コンピュート	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とする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ー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ー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. クラスター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ーテストは指定されたノード数で実行されます。

8.2.2. 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)/IOS	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
コンピュート	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とする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ー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ー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. クラスターは反復でスケーリングされます。

8.2.3. 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
コンピュート [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 つのコンピュート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、アイドリング 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 つのコンピュート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 つのコンピュート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ーの物理的な数は、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。

8.3. テスト済みのクラスター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ライブすると、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。メモリー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ー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。

```
required pods per cluster / pods per node = total number of nodes needed
```

ノードあたりの現在の Pod 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、

```
required pods per cluster / total number of nodes = expected pods per node
```

8.4. 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 (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。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 KiB** 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ルされる各 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9章 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。

9.1. 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9.1 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	● 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	● 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	● 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ー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。

重要

現時点で、CNS は OpenShift Container Platform 4.10 ではサポートされていません。

9.2. 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 クラスター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9.2 設定可能な推奨ストレージ技術

ストレージタイプ	ROX[1]	RWX[2]	レジストリー	スケーリングされたレジストリー	メトリック[3]	ロギング	アプリ
ブロック	はい[4]	いいえ	設定可能	設定不可	推奨	推奨	推奨
ファイル	はい[4]	○	設定可能	設定可能	設定可能[5]	設定可能[6]	推奨
オブジェクト	○	○	推奨	推奨	設定不可	設定不可	設定不可[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

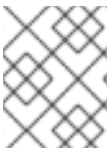
[3] **Prometheus はメトリックに使用される基礎となるテクノロジーです。**

[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、メトリック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] **ロギングの場合、共有ストレージを使用することはアンチパターンとなります。elasticsearch ごとに 1 つのボリュームが必要です。**

[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。

9.2.1. 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。

9.2.1.1. 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。

9.2.1.2. 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。

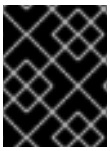
- ストレージ技術は、RWX アクセスモードをサポートする必要があります。
- ストレージ技術は、リードアフターライト (Read-After-Write) の一貫性を確保する必要があります。
- 推奨されるストレージ技術はオブジェクトストレージです。
- Red Hat OpenShift Data Foundation (ODF)、Amazon Simple Storage Service (Amazon S3)、Google Cloud Storage (GCS)、Microsoft Azure Blob Storage、および OpenStack Swift がサポートされています。
- 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。

9.2.1.3. メトリック

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。

9.2.1.4. 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:

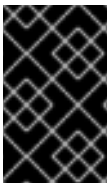
- 推奨されるストレージ技術はブロックストレージです。
- オブジェクトストレージは設定できません。

9.2.1.5. 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リング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。

9.2.2. 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スケ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。

9.3. 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9.3 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ーランタイム (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ーの容量によって制限されます。
/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ー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。**

9.4. 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。

第10章 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。

10.1. 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、ルートとイングレス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グレストラフィックのイングレスポイントです。

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

暗号化	LoadBalancerService	HostNetwork
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。

最新のプロセッサー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。

tuningOptions の詳細は、Ingress Controller 設定パラメーター 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。

第11章 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、コンテナーや 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。

11.1. ネットワークでの MTU の最適化

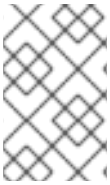
重要な Maximum Transmission Unit (MTU) が 2 つあります。1 つはネットワークインターフェイスコントローラー (NIC) MTU で、もう 1 つはクラスターネットワーク MTU です。

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) およびベアメタル環境は、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。

11.2. 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: OpenShiftSDN
  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。

11.3. 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 セキュリティー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、スループットの減少、CPU 使用率の上昇につながります。

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 デフォルト CNI ネットワークプロバイダーの設定パラメーター
- OpenShift SDN デフォルト CNI ネットワークプロバイダーの設定パラメーター

第12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。

12.1. 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。

12.2. 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を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

- コンピュートマシンのベアメタルへの追加

12.2.1. 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の名前を指定します。
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。

12.2.2. 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>  (1)
    disableCertificateVerification: True  (2)
  bootMACAddress: <host_boot_mac_address>
```

(1) **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。

(2) **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。

12.2.3. 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マシンセットにアノテーションを付けます。**<machineset>** を、マシンセット名に置き換えます。

```
$ 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す。

12.2.4. 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

- クラスターの拡張
- ベアメタル上の MachineHealthCheck

第13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

13.1. 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。トランスレーションルックアサイドバッファー (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。

13.2. 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ー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、これはスケジュール可能なリソース **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 (1)
      memory: "1Gi"
      cpu: "1"
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

(1) **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。コンテナーの分割は、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。

13.3. DOWNWARD API を使用した HUGE PAGE リソースの使用

Downward API を使用して、コンテナーで使用する 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。コンテナー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

- コンテナーによる Downward API オブジェクト使用の許可

13.4. 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。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。

13.4.1. ブート時

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ーが十分にある場合、**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。

13.5. 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一覧を確認します。

```
$ 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14章 低レイテンシーのノード向けの PERFORMANCE ADDON OPERATOR

14.1. 低レイテンシー

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ロールを果たします。

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スピードを決定します。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 5 G 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 50ms 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。コンピュート能力が文字通りクラウドのエッジ上にあり、ユーザーの近く置かれること考えてください。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、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するために自動チューニングを実装する Performance Addon Operator を提供します。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 コンテナー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ー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。

14.1.1. 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 プロセッサーコアが 2 つの論理コアとして機能することを可能にする Intel プロセッサーテクノロジーで、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

14.2. PERFORMANCE ADDON OPERATOR のインストール

Performance Addon Operator は、一連のノードで高度なノードのパフォーマンスチューニングを有効にする機能を提供します。クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して Performance Addon Operator をインストールできます。

14.2.1. CLI を使用した 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

- 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。
- 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. 以下のアクションを実行して、Performance Addon Operator の namespace を作成します。
   a. **openshift-performance-addon-operator** namespace を定義する以下の Namespace カスタムリソース (CR) を作成し、YAML を **pao-namespace.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-performance-addon-operator
  annotations:
    workload.openshift.io/allowed: management
```

   b. 以下のコマンドを実行して namespace を作成します。

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

2. 以下のオブジェクトを作成して、直前の手順で作成した namespace に Performance Addon Operator をインストールします。

   a. 以下の **OperatorGroup** CR を作成し、YAML を **pao-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-performance-addon-operator
  namespace: openshift-performance-addon-operator
```

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

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

   c. 以下のコマンドを実行して、次の手順に必要な **channel** の値を取得します。

```
$ oc get packagemanifest performance-addon-operator -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'
```

   出力例

```
4.10
```

   d. 以下の Subscription CR を作成し、YAML を **pao-sub.yaml** ファイルに保存します。

   Subscription の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-performance-addon-operator-subscription
  namespace: openshift-performance-addon-operator
spec:
  channel: "<channel>" (1)
  name: performance-addon-operator
  source: redhat-operators (2)
  sourceNamespace: openshift-marketplace
```

   (1) **.status.defaultChannel** パラメーターの直前の手順で取得した値を指定します。

   (2) **redhat-operators** 値を指定する必要があります。

   e. 以下のコマンドを実行して Subscription オブジェクトを作成します。

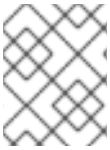
```
$ oc create -f pao-sub.yaml
```

   f. **openshift-performance-addon-operator** プロジェクトに切り替えます。

```
$ oc project openshift-performance-addon-operator
```

14.2.2. Web コンソールを使用した Performance Addon 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 Performance Addon Operator をインストールできます。

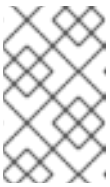


注記

先のセクションで説明されているように **Namespace** CR および **OperatorGroup** CR を作成する必要があります。

手順

1. OpenShift Container Platform Web コンソールを使用して Performance Addon 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 **Performance Addon Operator** を選択してから **Install** をクリックします。
   c. **Install Operator** ページで、**All namespaces on the cluster**を選択します。次に、**Install** をクリックします。
2. オプション: performance-addon-operator が正常にインストールされていることを確認します。
   a. **Operators → Installed Operators** ページに切り替えます。
   b. **Performance Addon Operator** が **openshift-operators** プロジェクトに **Succeeded** の **Status** でリストされていることを確認します。

   注記

   インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが成功し、**Succeeded** メッセージが表示された場合は、**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は、さらにトラブルシューティングを行うことができます。

   - **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、**openshift-operators** プロジェクトで Pod のログを確認します。

14.3. PERFORMANCE ADDON OPERATOR のアップグレード

次のマイナーバージョンの Performance Addon Operator に手動でアップグレードし、Web コンソールを使用して更新のステータスをモニターできます。

14.3.1. Performance Addon Operator のアップグレードについて

- OpenShift Container Platform Web コンソールを使用して Operator サブスクリプションのチャネルを変更することで、Performance Addon Operator の次のマイナーバージョンにアップグレードできます。
- Performance Addon Operator のインストール時に z-stream の自動更新を有効にできます。
- 更新は、OpenShift Container Platform のインストール時にデプロイされる Marketplace Operator 経由で提供されます。Marketplace Operator は外部 Operator をクラスターで利用可能にします。
- 更新の完了までにかかる時間は、ネットワーク接続によって異なります。ほとんどの自動更新は 15 分以内に完了します。

14.3.1.1. Performance Addon Operator のクラスターへの影響

- 低レイテンシーのチューニング Huge Page は影響を受けません。
- Operator を更新しても、予期しない再起動は発生しません。

14.3.1.2. Performance Addon Operator の次のマイナーバージョンへのアップグレード

OpenShift Container Platform Web コンソールを使用して Operator サブスクリプションのチャネルを変更することで、Performance Addon Operator を次のマイナーバージョンに手動でアップグレードできます。

前提条件

- cluster-admin ロールを持つユーザーとしてのクラスターへのアクセスがあること。

手順

1. Web コンソールにアクセスし、**Operators → Installed Operators** に移動します。
2. **Performance Addon Operator** をクリックし、**Operator details** ページを開きます。
3. **Subscription** タブをクリックし、**Subscription details** ページを開きます。
4. **Update channel** ペインで、バージョン番号の右側にある鉛筆アイコンをクリックし、**Change Subscription update channel** ウィンドウを開きます。
5. 次のマイナーバージョンを選択します。たとえば、Performance Addon Operator 4.10 にアップグレードする場合は、**4.10** を選択します。
6. **Save** をクリックします。
7. **Operators → Installed Operators** に移動してアップグレードのステータスを確認します。以下の **oc** コマンドを実行してステータスを確認することもできます。

```
$ oc get csv -n openshift-performance-addon-operator
```

14.3.1.3. 以前に特定の namespace にインストールされている場合の Performance Addon Operator のアップグレード

Performance Addon Operator をクラスターの特定の namespace(例: **openshift-performance-addon-operator**) にインストールしている場合、**OperatorGroup** オブジェクトを変更して、アップグレード前に **targetNamespaces** エントリーを削除します。

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 OpenShift クラスターにログインします。

手順

1. Performance Addon Operator **OperatorGroup** CR を編集し、以下のコマンドを実行して **targetNamespaces** エントリーが含まれる **spec** 要素を削除します。

```
$ oc patch operatorgroup -n openshift-performance-addon-operator openshift-performance-addon-operator --type json -p '[{ "op": "remove", "path": "/spec" }]'
```

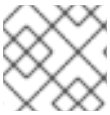
2. Operator Lifecycle Manager (OLM) が変更を処理するまで待機します。
3. OperatorGroup CR の変更が正常に適用されていることを確認します。**OperatorGroup** CR の **spec** 要素が削除されていることを確認します。

```
$ oc describe -n openshift-performance-addon-operator og openshift-performance-addon-operator
```

4. Performance Addon Operator のアップグレードに進みます。

14.3.2. アップグレードステータスの監視

Performance Addon Operator アップグレードステータスをモニターする最適な方法として、**ClusterServiceVersion** (CSV) **PHASE** を監視できます。Web コンソールを使用するか、**oc get csv** コマンドを実行して CSV の状態をモニターすることもできます。



注記

PHASE および状態の値は利用可能な情報に基づく近似値になります。

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**) がインストールされている。

手順

1. 以下のコマンドを実行します。

```
$ oc get csv
```

2. 出力を確認し、**PHASE** フィールドをチェックします。以下に例を示します。

```
VERSION    REPLACES                                 PHASE
4.10.0     performance-addon-operator.v4.10.0       Installing
4.8.0                                               Replacing
```

3. **get csv** を再度実行して出力を確認します。

```
# oc get csv
```

出力例

```
NAME                         DISPLAY                VERSION   REPLACES
PHASE
performance-addon-operator.v4.10.0   Performance Addon Operator   4.10.0    performance-
addon-operator.v4.8.0   Succeeded
```

14.4. 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。このような固有の要件を持つ業界では、OpenShift Container Platform は Performance Addon Operator を提供して、OpenShift Container Platform アプリケーションの低レイテンシーのパフォーマンスと一貫性のある応答時間を実現するための自動チューニングを実装します。

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) に更新するかどうかを指定し、Pod の infra コンテナー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ー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。

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。

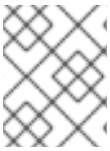
14.4.1. リアルタイムの既知の制限

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、コンテナー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。

14.4.2. 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. Performance Addon Operator をクラスターにインストールします。
2. オプション: ノードを OpenShift Container Platform クラスターに追加します。BIOS パラメーターの設定 について参照してください。
3. **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。
4. 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。

5. 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。Performance Addon Operator は、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想定します。すべてのノードを使用する場合は、Performance Addon 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。

6. 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。

7. **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
```

8. 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
```

9. 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が、ノードごとに 20 分の時間がかかることが予想されます。

10. すべてが予想通りに機能していることを確認します。

14.4.3. 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
```

文字列 **4.18.0-305.30.1.rt7.102.el8_4.x86_64 cri-o://1.23.0-99.rhaos4.10.gitc3131de.el8** 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   v1.23.0
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://1.23.0-99.rhaos4.10.gitc3131de.el8
[...]
```

14.4.4. 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. 適切なノードセレ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。

14.4.5. 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ー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ーには 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 つのコンテナーを持つ Pod の設定ファイルを示しています。コンテナー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。コンテナーには CPU 制限と CPU 要求があり、どちらも 1 CPU 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

コンテナー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、コンテナーが独自の 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。

14.4.6. 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
```

- Pod には **cpu-load-balancing.crio.io: true** アノテーションが必要です。

Performance Addon Operator は、該当するノードで高パフォーマンスのランタイムハンドラー設定スニペットの作成や、クラスターで高パフォーマンスのランタイムクラスの作成を行います。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ーのパフォーマンスに影響する可能性があります。

14.4.7. 適切なノードセレクターの割り当て

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レ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: ""
```

ノードセレクターの詳細は、Placing pods on specific nodes using node selectors を参照してください。

14.4.8. 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

Performance Addon Operator によって低レイテンシーを確保するために設定されたマシン設定プールに割り当てられるノードに一致するラベルセレ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。

14.4.9. Guaranteed Pod の分離された CPU のデバイス割り込み処理の管理

Performance Addon Operator は、Pod Infra コンテナーなど、予約された CPU を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タスクに、分離された CPU をワークロード実行用のアプリケーションコンテナーに分割して、ホストの CPU を管理できます。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

ローバルに無効にされません。デフォルトで、Performance Addon Operator は分離された CPU で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。

14.4.9.1. 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: performance.openshift.io/v2
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ーのパフォーマンスに影響を与える可能性があります。

14.4.9.2. Performance Addon Operator でのグローバルデバイス割り込み処理の無効化

Performance Addon Operator を分離された 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
...
```

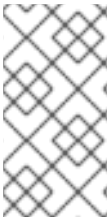
14.4.9.3. 個別の Pod の割り込み処理の無効化

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>
...
```

14.4.10. 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

Performance Addon 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。

14.4.10.1. サポート対象の API バージョン

Performance Addon 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。

14.4.10.1.1. Performance Addon Operator の v1alpha1 から v1 へのアップグレード

Performance Addon 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即時にオンザフライで変換され、API バージョン v1 の Performance Addon Operator に送信されます。

14.4.10.1.2. Performance Addon Operator API の v1alpha1 または v1 から v2 へのアップグレード

古い Performance Addon 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。

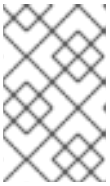
14.4.11. IRQ 動的負荷分散用ノードの設定

IRQ 動的負荷分散を処理するクラスターノードを設定するには、以下を実行します。

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 コンテナーは予約された CPU を使用し、アプリケーションコンテナーは分離された 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:v4.10"
    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。そのノードに対して SSH を実行し、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 アフィニティーマスクに **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

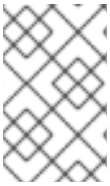
```
/proc/irq/0/smp_affinity_list: 0-5
/proc/irq/1/smp_affinity_list: 5
/proc/irq/2/smp_affinity_list: 0-5
/proc/irq/3/smp_affinity_list: 0-5
/proc/irq/4/smp_affinity_list: 0
/proc/irq/5/smp_affinity_list: 0-5
/proc/irq/6/smp_affinity_list: 0-5
/proc/irq/7/smp_affinity_list: 0-5
/proc/irq/8/smp_affinity_list: 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
```

一部の IRQ コントローラーは IRQ リバランスをサポートせず、常にすべてのオンライン CPU を IRQ マスクとして公開します。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。ホスト設定についての詳細は、ホストに対して SSH を実行し、**<irq-num>** をクエリーする CPU 番号に置き換えて以下を実行して参照してください。

```
$ cat /proc/irq/<irq-num>/effective_affinity
```

14.4.12. 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. 設定する必要のあるホストのどの 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
```

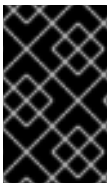
2. **PerformanceProfile** YAML で分離された CPU および予約された CPU を適用します。たとえば、論理コア CPU0 と CPU4 を**isolated** として設定し、論理コア CPU1 から CPU3 および CPU5 から CPU7 を**reserved** として設定できます。予約および分離された CPU を設定する場

合に、Pod 内の infra コンテナーは予約された CPU を使用し、アプリケーションコンテナーは分離された 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 プロセッサー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。

14.4.12.1. 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 コンテナーは予約された CPU を使用し、アプリケーションコンテナーは分離された CPU を使用します。

14.5. 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ーで使用します。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 コンテナーで使用します。

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。

14.5.1. Huge Page の設定

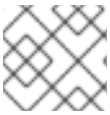
ノードは、OpenShift Container Platform クラスターで使用される Huge Page を事前に割り当てる必要があります。Performance Addon Operator を使用し、特定のノードで Huge Page を割り当てます。

OpenShift Container Platform は、Huge Page を作成し、割り当てる方法を提供します。Performance Addon Operator は、パーマンスプロファイルを使用してこれを実行するための簡単な方法を提供します。

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-###:  ###
```

14.5.2. 複数の Huge Page サイズの割り当て

同じコンテナーで異なるサイズの Huge Page を要求できます。これにより、Huge Page サイズのニーズの異なる複数のコンテナー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、Performance Addon Operator は以下に示すようにノード上に両方のサイズを設定できます。

```
spec:
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 1024
      node: 0
      size: 2M
    - count: 4
      node: 1
      size: 1G
```

14.5.3. infra およびアプリケーションコンテナーの 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、コンテナーランタイムはすべてのオンライン CPU を使用して、すべてのコンテナー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、Performance Add-On 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14.1 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。

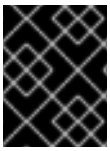
使用する正確なパーティショニングパターンは、ハードウェア、ワークロードの特性、予想されるシステム負荷などの多くの要因によって異なります。いくつかのサンプルユースケースは次のとおりです。

- 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。4.10 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ーワークロードの 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ー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タスクを実行する infra コンテナーの CPU を指定します。

(2) アプリケーションコンテナー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

14.6. PERFORMANCE ADDON OPERATOR を使用した NIC キューの削減

Performance Addon Operator を使用すると、パフォーマンスプロファイルを設定して、各ネットワークデバイスの Network Interface Card (NIC) キュー数を調整できます。デバイスネットワークキューを使用すると、パケットを複数の異なる物理キューに分散でき、各キューはパケット処理用に個別のスレッドを取得します。

リアルタイムまたは低レイテンシーシステムでは、分離 CPU にピニングされる不要な割り込み要求の行 (IRQ) をすべて予約またはハウスキーピング CPU に移動する必要があります。

OpenShift Container Platform ネットワークなど、システムが必要なアプリケーションのデプロイメントにおいて、または Data Plane Development Kit (DPDK) ワークロードを使用する混在型のデプロイメントにおいて、適切なスループットを実現するには複数のキューが必要であり、NIC キュー数は調整するか、変更しないようにする必要があります。たとえば、レイテンシーを低くするには、DPDK ベースのワークロードの NIC キューの数を、予約またはハウスキーピング CPU の数だけに減らす必要があります。

デフォルトでは CPU ごとに過剰なキューが作成されるので、チューニングしてレイテンシーを低くすると CPU のハウスキーピング向けの中断テーブルに収まりません。キューの数を減らすことで、適切なチューニングが可能になります。キューの数が少ないと、IRQ テーブルに適合する割り込みの数が少なくなります。

14.6.1. 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、Performance Addon 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。

     - 9**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。デバイスエントリー **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: 3-51,54-103
```

```
    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: 3-51,54-103
    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: 3-51,54-103
    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: 3-51,54-103
```

```
    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: 3-51,54-103
    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

14.6.2. 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。

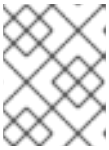
例 2

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

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、デバイスは以下のようになります。

```
# 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。

14.6.3. 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
```

14.7. 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。

Performance Addon Operator には **performanceProfile.spec.status.Conditions** ステータスフィールドが含まれます。

```
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)。

14.7.1. 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アドオンが作成するマシン設定の適用についての進捗に関する貴重な情報を保持します。パフォーマンスアドオンコントローラーは 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 がパフォーマンスプロファイルのステータスに返す状態は、MCP が **Degraded** の場合のみとなり、この場合、**performaceProfile.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
```

14.8. 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。

14.8.1. 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。

14.8.2. 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。

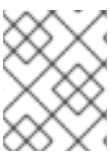
- Performance Addon Operator namespace および子オブジェクト
- **MachineConfigPool** および関連付けられた **MachineConfig** オブジェクト
- Node Tuning Operator および関連付けられた Tuned オブジェクト
- Linux カーネルコマンドラインオプション
- CPU および NUMA トポロジー
- 基本的な PCI デバイス情報と NUMA 局所性

must-gatherを使用して Performance Addon Operator のデバッグ情報を収集するには、Performance Addon Operator の**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:v4.10.
```

14.8.3. 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。

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 Performance Addon 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 \ 1

 --image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.10
2
```

1 デフォルトの OpenShift Container Platform **must-gather** イメージ。

2 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/ 1
```

1 **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 についての詳細は、ノードチューニング Operator について を参照してください。
- PerformanceProfile についての詳細は、Huge Page の設定 を参照してください。
- コンテナーからの Huge Page の消費に関する詳細は、Huge Page がアプリケーションによって消費される仕組み を参照してください。

第15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 コンテナーイメージは、**registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10** 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。

15.1. 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。

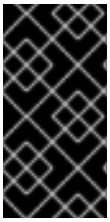
1. Performance Addon Operator を使用してパフォーマンスプロファイルを設定しました。
2. 必要なすべての CNF 設定をクラスターに適用しました。
3. クラスターに既存の **MachineConfigPool** CR が適用されている。デフォルトのワーカープールは **worker-cnf** です。

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、リアルタイムおよび低待機時間のワークロードのプロビジョニング を参照してください。

15.2. レイテンシーテストの検出モードについて

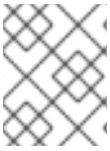
検出モードでは、設定を変更せずにクラスターの機能を検証できます。既存の環境設定はテストに使用されます。テストは、必要な設定アイテムを見つけ、それらのアイテムを使用してテストを実行できます。特定のテストの実行に必要なリソースが見つからない場合、テストは省略され、ユーザーに適切なメッセージが表示されます。テストが完了すると、事前に設定された設定項目のクリーンアップは行われず、テスト環境は別のテストの実行にすぐに使用できます。



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。

15.3. 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

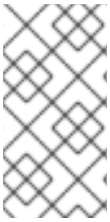
CPU 集約型 DPDK アプリケーションと同様に動作し、CPU の高いデータ処理をシミュレーションするビジーループにすべての中断と中断を測定します。

テストでは、次の環境変数が導入されます。

表15.1 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。

15.4. 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)/:/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。

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、実行中のコンテナーがコンテナー内から **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
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/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。

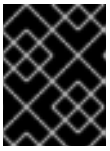
4. オプション: **-ginkgo.v** を追加して、詳細度を上げてテストを実行します。

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 LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
-e PERF_TEST_PROFILE=<performance_profile> registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.10 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

ここでは、以下のようになります。

<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。

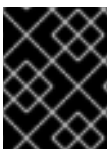


重要

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。

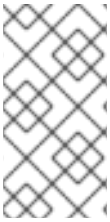
15.4.1. hwlatdetect の実行

hwlatdetect ツールは、Red Hat Enterprise Linux (RHEL) 8.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)/:/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

- 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。
- カスタマーポータルの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)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e
ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/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 µs) 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/validationsuite -ginkgo.v -ginkgo.focus=hwlatdetect
I0210 17:08:38.607699       7 request.go:668] Waited for 1.047200253s due to client-side
throttling, not priority and fairness, request:
GET:https://api.ocp.demo.lab:6443/apis/apps.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e validation
==========================================
Random Seed: 1644512917
Will run 0 of 48 specs

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
Ran 0 of 48 Specs in 0.001 seconds
SUCCESS! -- 0 Passed | 0 Failed | 0 Pending | 48 Skipped

PASS
Discovery mode enabled, skipping setup
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0210 17:08:41.179269      40 request.go:668] Waited for 1.046001096s due to client-side
throttling, not priority and fairness, request:
GET:https://api.ocp.demo.lab:6443/apis/storage.k8s.io/v1beta1?timeout=32s
```

```
Running Suite: CNF Features e2e integration tests
==================================================
Random Seed: 1644512920
Will run 1 of 151 specs

SSSSSSS
------------------------------
[performance] Latency Test with the hwlatdetect image
  should succeed
  /remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:221
STEP: Waiting two minutes to download the latencyTest image
STEP: Waiting another two minutes to give enough time for the cluster to move the pod to
Succeeded phase
Feb 10 17:10:56.045: [INFO]: found mcd machine-config-daemon-dzpw7 for node ocp-
worker-0.demo.lab
Feb 10 17:10:56.259: [INFO]: found mcd machine-config-daemon-dzpw7 for node ocp-
worker-0.demo.lab
Feb 10 17:11:56.825: [ERROR]: timed out waiting for the condition

• Failure [193.903 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:60
  with the hwlatdetect image
  /remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:213
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:221

    Log file created at: 2022/02/10 17:08:45
    Running on machine: hwlatdetect-cd8b6
    Binary: Built with gc go1.16.6 for linux/amd64
    Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
    I0210 17:08:45.716288       1 node.go:37] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-
56fabc639a679b757ebae30e5f01b2ebd38e9fde9ecae91c41be41d3e89b37f8/vmlinuz-
4.18.0-305.34.2.rt7.107.el8_4.x86_64 random.trust_cpu=on console=tty0
console=ttyS0,115200n8 ignition.platform.id=qemu
ostree=/ostree/boot.0/rhcos/56fabc639a679b757ebae30e5f01b2ebd38e9fde9ecae91c41be41d
3e89b37f8/0 root=UUID=56731f4f-f558-46a3-85d3-d1b579683385 rw rootflags=prjquota
skew_tick=1 nohz=on rcu_nocbs=3-5 tuned.non_isolcpus=ffffffc7 intel_pstate=disable
nosoftlockup tsc=nowatchdog intel_iommu=on iommu=pt isolcpus=managed_irq,3-5
systemd.cpu_affinity=0,1,2,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
29,30,31 + +
    I0210 17:08:45.716782       1 node.go:44] Environment information: kernel version 4.18.0-
305.34.2.rt7.107.el8_4.x86_64
    I0210 17:08:45.716861       1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 10 --window
10000000us --width 950000us]
    F0210 17:08:56.815204       1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect:  test duration 10 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: 24us 2
    Samples recorded: 1
    Samples exceeding threshold: 1
    ts: 1644512927.163556381, inner:20, outer:24
    ; err: exit status 1
    goroutine 1 [running]:
    k8s.io/klog.stacks(0xc000010001, 0xc00012e000, 0x25b, 0x2710)
        /remote-source/app/vendor/k8s.io/klog/klog.go:875 +0xb9
    k8s.io/klog.(*loggingT).output(0x5bed00, 0xc000000003, 0xc0000121c0, 0x53ea81, 0x7,
0x35, 0x0)
        /remote-source/app/vendor/k8s.io/klog/klog.go:829 +0x1b0
    k8s.io/klog.(*loggingT).printf(0x5bed00, 0x3, 0x5082da, 0x33, 0xc000113f58, 0x2, 0x2)
        /remote-source/app/vendor/k8s.io/klog/klog.go:707 +0x153
    k8s.io/klog.Fatalf(...)
        /remote-source/app/vendor/k8s.io/klog/klog.go:1276
    main.main()
        /remote-source/app/cnf-tests/pod-utils/hwlatdetect-runner/main.go:53 +0x897

    goroutine 6 [chan receive]:
    k8s.io/klog.(*loggingT).flushDaemon(0x5bed00)
        /remote-source/app/vendor/k8s.io/klog/klog.go:1010 +0x8b
    created by k8s.io/klog.init.0
        /remote-source/app/vendor/k8s.io/klog/klog.go:411 +0xd8

    goroutine 7 [chan receive]:
    k8s.io/klog/v2.(*loggingT).flushDaemon(0x5bede0)
        /remote-source/app/vendor/k8s.io/klog/v2/klog.go:1169 +0x8b
    created by k8s.io/klog/v2.init.0
        /remote-source/app/vendor/k8s.io/klog/v2/klog.go:420 +0xdf
    Unexpected error:
        <*errors.errorString | 0xc000418ed0>: {
            s: "timed out waiting for the condition",
        }
        timed out waiting for the condition
    occurred

    /remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:433
------------------------------
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSS
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-kni/performance-addon-
```

```
operators/functests/4_latency/latency.go:433

Ran 1 of 151 Specs in 222.254 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 150 Skipped

--- FAIL: TestTest (222.45s)
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。

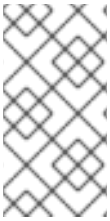
すべての遅延スパイクがハードウェアに関連しているわけではありません。ワークロードの要件を満たすようにホストファームウェアを調整してください。詳細は、システムチューニング用のファームウェアパラメーターの設定 を参照してください。

15.4.2. cyclictest の実行

cyclictest 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)/:/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

- カスタマーポータルの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。

- Performance アドオンオペレーターを使用して、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)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -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:v4.10 \
/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 µs) 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

```
Discovery mode enabled, skipping setup
running /usr/bin//cnftests -ginkgo.v -ginkgo.focus=cyclictest
I0811 15:02:36.350033      20 request.go:668] Waited for 1.049965918s due to client-side
throttling, not priority and fairness, request:
GET:https://api.cnfdc8.t5g.lab.eng.bos.redhat.com:6443/apis/machineconfiguration.openshift.io/
v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=================================================
Random Seed: 1628694153
Will run 1 of 138 specs

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
------------------------------
[performance] Latency Test with the cyclictest image
  should succeed
  /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:200
STEP: Waiting two minutes to download the latencyTest image
STEP: Waiting another two minutes to give enough time for the cluster to move the pod to
Succeeded phase
Aug 11 15:03:06.826: [INFO]: found mcd machine-config-daemon-wf4w8 for node
cnfdc8.clus2.t5g.lab.eng.bos.redhat.com

• Failure [22.527 seconds]
[performance] Latency Test
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
```

```
kni/performance-addon-operators/functests/4_latency/latency.go:84
  with the cyclictest image
  /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:188
    should succeed [lt]
    /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:200

    The current latency 27 is bigger than the expected one 20
    Expected
        <bool>: false
    to be true

    /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:219

Log file created at: 2021/08/11 15:02:51
Running on machine: cyclictest-knk7d
Binary: Built with gc go1.16.6 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0811 15:02:51.092254       1 node.go:37] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-
612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a543/vmlinuz-
4.18.0-305.10.2.rt7.83.el8_4.x86_64 ip=dhcp random.trust_cpu=on console=tty0
console=ttyS0,115200n8
ostree=/ostree/boot.1/rhcos/612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe11565497
1aa58a543/0 ignition.platform.id=openstack root=UUID=5a4ddf16-9372-44d9-ac4e-
3ee329e16ab3 rw rootflags=prjquota skew_tick=1 nohz=on rcu_nocbs=1-3
tuned.non_isolcpus=000000ff,ffffffff,ffffffff,ffffffff1 intel_pstate=disable nosoftlockup
tsc=nowatchdog intel_iommu=on iommu=pt isolcpus=managed_irq,1-3
systemd.cpu_affinity=0,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
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,
59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,
89,90,91,92,93,94,95,96,97,98,99,100,101,102,103 default_hugepagesz=1G
hugepagesz=2M hugepages=128 nmi_watchdog=0 audit=0 mce=off
processor.max_cstate=1 idle=poll intel_idle.max_cstate=0
I0811 15:02:51.092427       1 node.go:44] Environment information: kernel version 4.18.0-
305.10.2.rt7.83.el8_4.x86_64
I0811 15:02:51.092450       1 main.go:48] running the cyclictest command with arguments \
[-D 600 -95 1 -t 10 -a 2,4,6,8,10,54,56,58,60,62 -h 30 -i 1000 --quiet]
I0811 15:03:06.147253       1 main.go:54] succeeded to run the cyclictest command: #
/dev/cpu_dma_latency set to 0us
# Histogram
000000 000000   000000  000000  000000  000000  000000  000000  000000  000000
000000
000001 000000   005561  027778  037704  011987  000000  120755  238981  081847
300186
000002 587440   581106  564207  554323  577416  590635  474442  357940  513895
296033
000003 011751   011441  006449  006761  008409  007904  002893  002066  003349
003089
000004 000527   001079  000914  000712  001451  001120  000779  000283  000350
000251

More histogram entries ...
# Min Latencies: 00002 00001 00001 00001 00001 00002 00001 00001 00001 00001
```

```
# Avg Latencies: 00002 00002 00002 00001 00002 00002 00001 00001 00001 00001
# Max Latencies: 00018 00465 00361 00395 00208 00301 02052 00289 00327 00114
# Histogram Overflows: 00000 00220 00159 00128 00202 00017 00069 00059 00045
00120
# Histogram Overflow at cycle number:
# Thread 0:
# Thread 1: 01142 01439 05305 … # 00190 others
# Thread 2: 20895 21351 30624 … # 00129 others
# Thread 3: 01143 17921 18334 … # 00098 others
# Thread 4: 30499 30622 31566 ... # 00172 others
# Thread 5: 145221 170910 171888 ...
# Thread 6: 01684 26291 30623 ...# 00039 others
# Thread 7: 28983 92112 167011 … 00029 others
# Thread 8: 45766 56169 56171 ...# 00015 others
# Thread 9: 02974 08094 13214 ... # 00090 others
```

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、18µs 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: cyclictest -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  000000
000001 000000   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: cyclictest -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  000000
000001 000000   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
# Avg Latencies: 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
```

15.4.3. 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)/:/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

- カスタマーポータルの認証情報を使用して、**registry.redhat.io** にログインしました。
- Performance アドオンオペレーターを使用して、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)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e
ROLE_WORKER_CNF=worker-cnf \
-e LATENCY_TEST_CPUS=7 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/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 µs) 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//validationsuite -ginkgo.v -ginkgo.focus=oslat
I0829 12:36:55.386776       8 request.go:668] Waited for 1.000303471s due to client-side
throttling, not priority and fairness, request:
GET:https://api.cnfdc8.t5g.lab.eng.bos.redhat.com:6443/apis/authentication.k8s.io/v1?
timeout=32s
Running Suite: CNF Features e2e validation
==========================================

Discovery mode enabled, skipping setup
running /usr/bin//cnftests -ginkgo.v -ginkgo.focus=oslat
```

```
I0829 12:37:01.219077      20 request.go:668] Waited for 1.050010755s due to client-side
throttling, not priority and fairness, request:
GET:https://api.cnfdc8.t5g.lab.eng.bos.redhat.com:6443/apis/snapshot.storage.k8s.io/v1beta1?
timeout=32s
Running Suite: CNF Features e2e integration tests
==================================================
Random Seed: 1630240617
Will run 1 of 142 specs

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
------------------------------
[performance] Latency Test with the oslat image
  should succeed
  /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:134
STEP: Waiting two minutes to download the latencyTest image
STEP: Waiting another two minutes to give enough time for the cluster to move the pod to
Succeeded phase
Aug 29 12:37:59.324: [INFO]: found mcd machine-config-daemon-wf4w8 for node
cnfdc8.clus2.t5g.lab.eng.bos.redhat.com

• Failure [49.246 seconds]
[performance] Latency Test
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:59
  with the oslat image
  /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:112
    should succeed [It]
    /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:134

    The current latency 27 is bigger than the expected one 20 (1)
    Expected
        <bool>: false
    to be true
 /go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-
kni/performance-addon-operators/functests/4_latency/latency.go:168

Log file created at: 2021/08/29 13:25:21
Running on machine: oslat-57c2g
Binary: Built with gc go1.16.6 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0829 13:25:21.569182       1 node.go:37] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-
612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a543/vmlinuz-
4.18.0-305.10.2.rt7.83.el8_4.x86_64 ip=dhcp random.trust_cpu=on console=tty0
console=ttyS0,115200n8
ostree=/ostree/boot.0/rhcos/612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe11565497
1aa58a543/0 ignition.platform.id=openstack root=UUID=5a4ddf16-9372-44d9-ac4e-
3ee329e16ab3 rw rootflags=prjquota skew_tick=1 nohz=on rcu_nocbs=1-3
tuned.non_isolcpus=000000ff,ffffffff,ffffffff,ffffffff1 intel_pstate=disable nosoftlockup
tsc=nowatchdog intel_iommu=on iommu=pt isolcpus=managed_irq,1-3
systemd.cpu_affinity=0,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,
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,
59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,
```

```
89,90,91,92,93,94,95,96,97,98,99,100,101,102,103 default_hugepagesz=1G
hugepagesz=2M hugepages=128 nmi_watchdog=0 audit=0 mce=off
processor.max_cstate=1 idle=poll intel_idle.max_cstate=0
I0829 13:25:21.569345       1 node.go:44] Environment information: kernel version 4.18.0-
305.10.2.rt7.83.el8_4.x86_64
I0829 13:25:21.569367       1 main.go:53] Running the oslat command with arguments \
[--duration 600 --rtprio 1 --cpu-list 4,6,52,54,56,58 --cpu-main-thread 2]
I0829 13:35:22.632263       1 main.go:59] Succeeded to run the oslat command: oslat V 2.00
Total runtime:    600 seconds
Thread priority:  SCHED_FIFO:1
CPU list:     4,6,52,54,56,58
CPU for main thread:  2
Workload:     no
Workload mem:     0 (KiB)
Preheat cores:    6

Pre-heat for 1 seconds...
Test starts...
Test completed.

       Core:  4 6 52 54 56 58
   CPU Freq:  2096 2096 2096 2096 2096 2096 (Mhz)
   001 (us):  19390720316 19141129810 20265099129 20280959461 19391991159
19119877333
   002 (us):  5304 5249 5777 5947 6829 4971
   003 (us):  28 14 434 47 208 21
   004 (us):  1388 853 123568 152817 5576 0
   005 (us):  207850 223544 103827 91812 227236 231563
   006 (us):  60770 122038 277581 323120 122633 122357
   007 (us):  280023 223992 63016 25896 214194 218395
   008 (us):  40604 25152 24368 4264 24440 25115
   009 (us):  6858 3065 5815 810 3286 2116
   010 (us):  1947 936 1452 151 474 361
 ...
    Minimum:  1 1 1 1 1 1 (us)
    Average:  1.000 1.000 1.000 1.000 1.000 1.000 (us)
    Maximum:  37 38 49 28 28 19 (us)
    Max-Min:  36 37 48 27 27 18 (us)
   Duration:  599.667 599.667 599.667 599.667 599.667 599.667 (sec)
```

1 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。

15.5. 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)/:/kubeconfig:Z -v $(pwd)/reportdest:<report_folder_path> \
-e KUBECONFIG=/kubeconfig/kubeconfig  -e DISCOVERY_MODE=true \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/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。

15.6. 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)/:/kubeconfig:Z -v $(pwd)/junitdest:<junit_folder_path> \
-e KUBECONFIG=/kubeconfig/kubeconfig  -e DISCOVERY_MODE=true \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/usr/bin/test-run.sh --junit <junit_folder_path> \
-ginkgo.focus="\[performance\]\ Latency\ Test"
```

ここでは、以下のようになります。

<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。

15.7. 単一ノードの OPENSHIFT クラスターでレイテンシーテストを実行する

単一ノードの OpenShift クラスター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)/:/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。

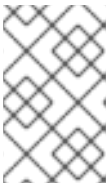
前提条件

- 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 クラスター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 ROLE_WORKER_CNF=master \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
/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。

15.8. 切断されたクラスター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:v4.10 \
/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 IMAGE_REGISTRY="<disconnected_registry>" \
-e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.10" \
/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>" \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 /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ー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 '.["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-rhel8:4.10 \
/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 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": "imageforcnftests:4.10"
  }
]
```

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ー内の **/kubeconfig** 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)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
```

```
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubeconfig/images.json" \
| oc image mirror -f -
```

15.9. CNF-TESTS コンテナー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** コンテナー内からクラスターにアクセスできる必要があります。

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

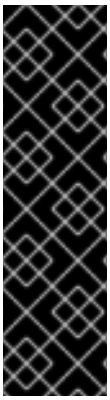
- 次のコマンドを実行して、**cnf-tests** コンテナー内からクラスターにアクセスできることを確認します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.10 \
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16章 クラスター更新のための TOPOLOGY AWARE LIFECYCLE MANAGER

TALM (Topology Aware Lifecycle Manager) を使用して、複数の単一ノード OpenShift クラスターのソフトウェアライフサイクルを管理することができます。TALM は Red Hat Advanced Cluster Management (RHACM) ポリシーを使用して、ターゲットクラスター上で変更を実行します。



重要

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

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、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. 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

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。

16.2. 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。

16.3. 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 regitry でハブクラスターを設定します。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. 利用可能な Operator のリストから **Topology Aware Lifecycle Manager** を検索し、**Install** をクリックします。
3. **Installation mode** ["All namespaces on the cluster (default)"] および **Installed Namespace** ("openshift-operators") のデフォルトの選択を維持し、Operator が適切にインストールされていることを確認します。
4. **Install** をクリックします。

検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators → Installed Operators** ページに移動します。
2. Operator が **All Namespaces** ネームスペースにインストールされ、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators → Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads → Pods** ページに移動し、問題を報告している **cluster-group-upgrades-controller-manager** Pod のコンテナーのログを確認します。

16.4. 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 regitry でハブクラスターを設定します。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **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.4.10.0-202206301927      Topology Aware Lifecycle
Manager   4.10.0-202206301927              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

出力例

```
NAMESPACE                              NAME                                        READY   UP-TO-
DATE   AVAILABLE   AGE
openshift-operators                    cluster-group-upgrades-controller-manager   1/1
1           1           14s
```

16.5. 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

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 は次の状態になる可能性があります。

- **UpgradeNotStarted**
- **UpgradeCannotStart**
- **UpgradeNotComplete**
- **UpgradeTimedOut**
- **UpgradeCompleted**
- **PrecachingRequired**

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

16.5.1. UpgradeNotStarted 状態

ClusterGroupUpgrade CR の初期状態は **UpgradeNotStarted** です。

TALM は以下のフィールドに基づいて修復計画をビルドします。

- **clusterSelector** 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。
- **clusters** フィールドは、更新するクラスターのリストを指定します。
- **canaries** フィールドは、カナリア更新のクラスターを指定します。
- **maxConcurrency** フィールドは、バッチで更新するクラスターの数を指定します。

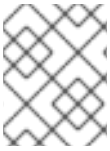
clusters フィールドと **clusterSelector** フィールドを一緒に使用して、結合されたクラスターのリストを作成できます。

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。

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。

ClusterGroupUpgrade CR は、修復計画が正常に作成され、**enable** フィールドが **true** に設定された後、**UpgradeNotCompleted** 状態に移行します。この時点で、TALM は指定されたマネージドクラスターでコンプライアンス違反のクラスターの更新を開始します。



注記

ClusterGroupUpgrade CR が **UpgradeNotStarted** または **UpgradeCannotStart** 状態の場合にのみ、**spec** フィールドを変更できます。

UpgradeNotStarted 状態の ClusterGroupUpgrade CR のサンプル

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters: (1)
  - spoke1
  enable: false
  managedPolicies: (2)
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy: (3)
    canaries: (4)
      - spoke1
    maxConcurrency: 1 (5)
    timeout: 240
status: (6)
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
```

```
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
managedPoliciesForUpgrade:
- name: policy1-common-cluster-version-policy
  namespace: default
- name: policy2-common-pao-sub-policy
  namespace: default
placementBindings:
- cgu-upgrade-complete-policy1-common-cluster-version-policy
- cgu-upgrade-complete-policy2-common-pao-sub-policy
placementRules:
- cgu-upgrade-complete-policy1-common-cluster-version-policy
- cgu-upgrade-complete-policy2-common-pao-sub-policy
remediationPlan:
- - spoke1
```

1. 更新するクラスターのリストを定義します。
2. 修正するユーザー定義のポリシーセットをリスト表示します。
3. クラスター更新の詳細を定義します。
4. カナリア更新のクラスターを定義します。
5. バッチの同時更新の最大数を定義します。修復バッチの数は、カナリアクラスターの数に加えて、カナリアクラスターを除くクラスターの数を **maxConcurrency** 値で除算します。すべての管理ポリシーに準拠しているクラスターは、修復計画から除外されます。
6. 更新のステータスに関する情報を表示します。

16.5.2. UpgradeCannotStart 状態

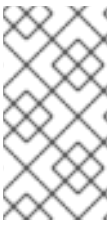
UpgradeCannotStart 状態では、以下の理由により更新を開始できません。

- システムに CR のブロックがない
- ブロッキング CR がまだ終了していない

16.5.3. UpgradeNotCompleted 状態

UpgradeNotCompleted 状態では、TALM は **UpgradeNotStarted** 状態で定義される修復計画に従ってポリシーを強制します。

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。



注記

管理されたポリシーは、**ClusterGroupUpgrade** CR の **managedPolicies** フィールドにリスト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。指定されたクラスターが現在のポリシーに準拠した後、次の管理ポリシーが次の準拠していないクラスターに適用されます。

UpgradeNotCompleted 状態の ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true (1)
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: (2)
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non compliant
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  placementBindings:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  placementRules:
  - cgu-upgrade-complete-policy1-common-cluster-version-policy
  - cgu-upgrade-complete-policy2-common-pao-sub-policy
  remediationPlan:
  - - spoke1
  status:
    currentBatch: 1
    remediationPlanForBatch: (3)
      spoke1: 0
```

(1) **spec.enable** フィールドの値が **true** になると、更新が開始されます。

(2) 更新が開始されると、それに応じて **status** フィールドが変化します。

(3) バッチ内のクラスターと、各クラスターに現在適用されているポリシーのインデックスをリスト表示します。ポリシーのインデックスは **0** で始まり、インデックスは **status.managedPoliciesForUpgrade** リストの順番に従います。

16.5.4. UpgradeTimedOut 状態

UpgradeTimedOut 状態で、TALM は **ClusterGroupUpgrade** CR のすべてのポリシーが準拠しているかどうかを 1 時間ごとに確認します。チェックは、**ClusterGroupUpgrade** CR が削除されるか、更新が

完了するまで継続されます。定期的なチェックにより、ネットワーク、CPU、またはその他の問題により発生する場合に更新が完了できます。

TALM は、2 つの場合に **UpgradeTimedOut** 状態に移行します。

- 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ない場合。
- クラスターが **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ない場合。

ポリシーが準拠している場合、TALM は **UpgradeCompleted** 状態に移行します。

16.5.5. UpgradeCompleted 状態

UpgradeCompleted 状態で、クラスターの更新が完了します。

UpgradeCompleted 状態の ClusterGroupUpgrade CR のサンプル

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  actions:
    afterCompletion:
      deleteObjects: true (1)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: (2)
  conditions:
  - message: The ClusterGroupUpgrade CR has all clusters compliant with all the managed policies
    reason: UpgradeCompleted
    status: "True"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  remediationPlan:
  - - spoke1
  status:
    remediationPlanForBatch:
      spoke1: -2 (3)
```

1. **spec.action.afterCompletion.deleteObjects** フィールドの値はデフォルトで **true** です。更新が完了すると、TALM は更新中に作成された基礎となる RHACM オブジェクトを削除します。このオプションは、更新が正常に完了した後に RHACM ハブがコンプライアンスを継続的にチェックしないようにするためです。

2. **status** フィールドは、更新が正常に完了したことを示しています。

3. すべてのポリシーがクラスターに適用されていることを表示します。

<discreet><title>PrecachingRequired 状態</title>
PrecachingRequired 状態の場合は、更新を開始する前に、クラスターにイメージを事前キャッシュする必要があります。事前キャッシュの詳細は、コンテナー イメージの事前キャッシュ機能の使用セクションを参照してください。

</discreet>

16.5.6. ClusterGroupUpgrade CR のブロック

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

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

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

前提条件

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

手順

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

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
```

```
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
```

1 ブロッキング CR を定義します。**cgu-c** が完了するまで **cgu-a** の更新を開始できません。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs: 1
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: false
  managedPolicies:
```

```
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

1 **cgu-a** が完了するまで **cgu-b** の更新を開始できません。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec: 1
  clusters:
  - spoke6
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
```

```
    - policy2-common-pao-sub-policy
    - policy3-common-ptp-sub-policy
    - policy4-common-sriov-sub-policy
    remediationStrategy:
      maxConcurrency: 1
      timeout: 240
  status:
    conditions:
    - message: The ClusterGroupUpgrade CR is not enabled
      reason: UpgradeNotStarted
      status: "False"
      type: Ready
    copiedPolicies:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    managedPoliciesCompliantBeforeUpgrade:
    - policy2-common-pao-sub-policy
    - policy3-common-ptp-sub-policy
    managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy4-common-sriov-sub-policy
      namespace: default
    placementBindings:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    placementRules:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    remediationPlan:
    - - spoke6
    status: {}
```

1. **cgu-c** の更新にはブロック CR がありません。TALM は、 **enable** フィールドが **true** に設定されている場合に **cgu-c** の更新を開始します。

2. 関連する CR ごとに以下のコマンドを実行して **ClusterGroupUpgrade** CR を作成します。

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

3. 関連する各 CR について以下のコマンドを実行して、更新プロセスを開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/<name> \
--type merge -p '{"spec":{"enable":true}}'
```

以下の例は、**enable** フィールドが **true** に設定されている **ClusterGroupUpgrade** CR を示しています。

ブロッキング CR のある cgu-a の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
```

```
  namespace: default
spec:
  blockingCRs:
  - name: cgu-c
    namespace: default
  clusters:
  - spoke1
  - spoke2
  - spoke3
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  remediationStrategy:
    canaries:
    - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-c]' (1)
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ptp-sub-policy
  remediationPlan:
  - - spoke1
  - - spoke2
  status: {}
```

(1) ブロッキング CR のリストを表示します。

ブロッキング CR のある cgu-b の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs:
  - name: cgu-a
    namespace: default
  clusters:
  - spoke4
  - spoke5
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-a]' (1)
    reason: UpgradeCannotStart
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ptp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ptp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}
```

-

1 ブロッキング CR のリストを表示します。

CR をブロックする cgu-c の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec:
  clusters:
  - spoke6
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non compliant
1
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status:
    currentBatch: 1
    remediationPlanForBatch:
      spoke6: 0
```

1 **cgu-c** の更新にはブロック CR がありません。

16.6. 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ーを追加します。次に、TALM はポリシーの **status.status** フィールドを確認し続けます。
- ポリシーの **status.status** がない場合、TALM はエラーを生成します。
- クラスターのコンプライアンスステータスがポリシーの **status.status** フィールドにない場合、TALM はそのクラスターをそのポリシーに準拠していないと見なします。

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、PolicyGenTemplate CRD について を参照してください。

16.6.1. マネージドクラスターへの更新ポリシーの適用

ポリシーを適用してマネージドクラスターを更新できます。

前提条件

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

手順

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

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
```

```
  managedPolicies: (1)
    - policy1-common-cluster-version-policy
    - policy2-common-pao-sub-policy
    - policy3-common-ptp-sub-policy
    - policy4-common-sriov-sub-policy
  enable: false
  clusters: (2)
  - spoke1
  - spoke2
  - spoke5
  - spoke6
  remediationStrategy:
    maxConcurrency: 2 (3)
    timeout: 240 (4)
```

(1) 適用するポリシーの名前。

(2) 更新するクラスターのリスト。

(3) **maxConcurrency** フィールドは、同時に更新されるクラスターの数を示します。

(4) 更新のタイムアウト (分単位)。

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
default     cgu-1     8m55s
```

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": "The ClusterGroupUpgrade CR is not enabled", (1)
      "reason": "UpgradeNotStarted",
      "status": "False",
      "type": "Ready"
    }
```

```
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-pao-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"performance-
addon-operator\",\"namespace\":\"openshift-performance-addon-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-operator-
subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-network-
operator-subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-pao-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-pao-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-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                                REMEDIATION ACTION
COMPLIANCE STATE   AGE
default     cgu-policy1-common-cluster-version-policy          enforce
17m 1
default     cgu-policy2-common-pao-sub-policy                  enforce
17m
default     cgu-policy3-common-ptp-sub-policy                  enforce
17m
default     cgu-policy4-common-sriov-sub-policy                enforce
17m
default     policy1-common-cluster-version-policy              inform            NonCompliant
15h
default     policy2-common-pao-sub-policy                      inform            NonCompliant
15h
default     policy3-common-ptp-sub-policy                      inform            NonCompliant
18m
default     policy4-common-sriov-sub-policy                    inform            NonCompliant
18m
```

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

3. 以下のコマンドを実行して、**spec.enable** フィールドの値を **true** に変更します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-1 \
--patch '{"spec":{"enable":true}}' --type=merge
```

検証

1. 以下のコマンドを実行して更新のステータスを再度確認します。

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

出力例

```
{
  "computedMaxConcurrency": 2,
  "conditions": [ 1
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant",
      "reason": "UpgradeNotCompleted",
      "status": "False",
      "type": "Ready"
    }
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-pao-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"performance-
addon-operator\",\"namespace\":\"openshift-performance-addon-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-operator-
subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-network-
operator-subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-pao-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-pao-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
```

```
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-pao-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "precaching": {
    "spec": {}
  },
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {
    "currentBatch": 1,
    "currentBatchStartedAt": "2022-02-25T15:54:16Z",
    "remediationPlanForBatch": {
      "spoke1": 0,
      "spoke2": 1
    },
    "startedAt": "2022-02-25T15:54:16Z"
  }
}
```


1 現在のバッチの更新の進捗を反映します。このコマンドを再度実行して、進捗に関する更新情報を取得します。

2. ポリシーに Operator サブスクリプションが含まれる場合、インストールの進捗を単一ノードクラスターで直接確認できます。

   a. 以下のコマンドを実行して、インストールの進捗を確認する単一ノードクラスターの **KUBECONFIG** ファイルをエクスポートします。

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

   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   4.9.5     True        True          43s     Working towards 4.9.7: 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。

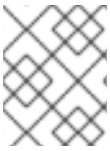
```
$ oc get csv -n <operator_namespace>
```

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
```

16.7. コンテナーイメージ事前キャッシュ機能の使用

クラスターにはコンテナーイメージレジストリーにアクセスするための帯域幅が制限されるため、更新が完了する前にタイムアウトが発生する可能性があります。



注記

更新の時間は TALM によって設定されていません。手動アプリケーションまたは外部自動化により、更新の開始時に **ClusterGroupUpgrade** CR を適用できます。

コンテナーイメージの事前キャッシュは、**ClusterGroupUpgrade** CR で **preCaching** フィールドが **true** に設定されている場合に起動します。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。

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。

PrecacheNotStarted

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。

PrecachePreparing

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。

PrecacheStarting

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。

PrecacheActive

ジョブは Active の状態です。

PrecacheSucceeded

キャッシュ前の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が部分的に行われました。

PrecacheUnrecoverableError

ジョブはゼロ以外の終了コードで終了します。

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

pre-cache 機能により、更新の開始前に、必要なコンテナーイメージをスポーククラスターに置くことができます。

前提条件

- Topology Aware Lifecycle Manager (TALM) をインストールします。
- 1 つ以上のマネージドクラスターをプロビジョニングします。

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

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

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true (1)
  clusters:
  - cnfdb1
  - cnfdb2
  enable: false
  managedPolicies:
  - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

(1) **preCaching** フィールドは **true** に設定されています。これにより、更新を開始する前に TALM がコンテナーイメージをプルできます。

2. 更新を開始する場合は、以下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
ztp-group-du-sno   du-upgrade-4918  10s (1)
```

(1) CR が作成されます。

2. 以下のコマンドを実行して、事前キャッシュタスクのステータスを確認します。

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

出力例

```
{
  "conditions": [
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is not completed (required)", 1
      "reason": "PrecachingRequired",
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is required and not done",
      "reason": "PrecachingNotDone",
      "status": "False",
      "type": "PrecachingDone"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:34Z",
      "message": "Pre-caching spec is valid and consistent",
      "reason": "PrecacheSpecIsWellFormed",
      "status": "True",
      "type": "PrecacheSpecValid"
    }
  ],
  "precaching": {
    "clusters": [
      "cnfdb1" 2
    ],
    "spec": {
      "platformImage": "image.example.io"},
    "status": {
      "cnfdb1": "Active"}
    }
}
```

1 更新中であることを表示します。

2 特定されたクラスターのリストを表示します。

3. スポーククラスターで以下のコマンドを実行して、事前キャッシュジョブのステータスを確認します。

```
$ oc get jobs,pods -n openshift-talm-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": "PrecachingDone" 1
   }
```

1 キャッシュ前のタスクが実行されます。

16.8. TOPOLOGY AWARE LIFECYCLE MANAGER のトラブルシューティング

Topology Aware Lifecycle Manager (TALM) は、RHACM ポリシーを修復する OpenShift Container Platform Operator です。問題が発生した場合には、**oc adm must-gather** コマンドを使用して詳細およびログを収集し、問題のデバッグ手順を行います。

関連トピックの詳細は、以下のドキュメントを参照してください。

- Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix
- Red Hat Advanced Cluster Management Troubleshooting
- Operator の問題のトラブルシューティングセクション

16.8.1. 一般的なトラブルシューティング

以下の質問を確認して、問題の原因を特定できます。

- 適用する設定がサポートされているか？
  - RHACM と OpenShift Container Platform のバージョンと互換性があるか？
  - TALM および RHACM のバージョンと互換性があるか？
- 問題の原因となる以下のコンポーネントはどれですか？
  - 「管理ポリシー」
  - 「クラスター」

- 「修復ストラテジー」
- 「Topology Aware Lifecycle Manager」

ClusterGroupUpgrade 設定が機能するようにするには、以下を実行できます。

1. **spec.enable** フィールドを **false** に設定して **ClusterGroupUpgrade** CR を作成します。
2. ステータスが更新され、トラブルシューティングの質問を確認するのを待ちます。
3. すべてが予想通りに機能する場合は、**ClusterGroupUpgrade** CR で **spec.enable** フィールドを **true** に設定します。

警告

ClusterUpgradeGroup CR で **spec.enable** フィールドを **true** に設定すると、更新手順が起動し、CR の **spec** フィールドを編集することができなくなります。

16.8.2. ClusterUpgradeGroup CR を変更できません。

問題

更新を有効にした後に、**ClusterUpgradeGroup** CR を編集することはできません。

解決方法

以下の手順を実行して手順を再起動します。

1. 以下のコマンドを実行して古い **ClusterGroupUpgrade** CR を削除します。

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. マネージドクラスターおよびポリシーに関する既存の問題を確認し、修正します。
   a. すべてのクラスターがマネージドクラスターで、利用可能であることを確認します。
   b. すべてのポリシーが存在し、**spec.remediationAction** フィールドが **inform** に設定されていることを確認します。
3. 正しい設定で新規の **ClusterGroupUpgrade** CR を作成します。

```
$ oc apply -f <ClusterGroupUpgradeCR_YAML>
```

16.8.3. 管理ポリシー

システムでの管理ポリシーの確認

問題

システムで正しい管理ポリシーがあるかどうかをチェックする。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

出力例

```
["group-du-sno-validator-du-validator-policy", "policy2-common-pao-sub-policy", "policy3-common-ptp-sub-policy"]
```

remediationAction モードの確認

問題

remediationAction フィールドが、管理ポリシーの **spec** で **inform** に設定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get policies --all-namespaces
```

出力例

```
NAMESPACE   NAME                                REMEDIATION ACTION   COMPLIANCE
STATE   AGE
default     policy1-common-cluster-version-policy          inform            NonCompliant
5d21h
default     policy2-common-pao-sub-policy                  inform            Compliant         5d21h
default     policy3-common-ptp-sub-policy                  inform            NonCompliant      5d21h
default     policy4-common-sriov-sub-policy                inform            NonCompliant      5d21h
```

ポリシーコンプライアンスの状態の確認

問題

ポリシーのコンプライアンス状態を確認する。

解決方法

以下のコマンドを実行します。

```
$ oc get policies --all-namespaces
```

出力例

```
NAMESPACE   NAME                                REMEDIATION ACTION   COMPLIANCE
STATE   AGE
default     policy1-common-cluster-version-policy          inform            NonCompliant
5d21h
default     policy2-common-pao-sub-policy                  inform            Compliant         5d21h
default     policy3-common-ptp-sub-policy                  inform            NonCompliant      5d21h
default     policy4-common-sriov-sub-policy                inform            NonCompliant      5d21h
```

16.8.4. クラスター

マネージドクラスターが存在するかどうかの確認

問題

ClusterGroupUpgrade CR のクラスターがマネージドクラスターかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED   AVAILABLE   AGE
local-cluster   true           https://api.hub.example.com:6443      True     Unknown     13d
spoke1          true           https://api.spoke1.example.com:6443   True     True        13d
spoke3          true           https://api.spoke3.example.com:6443   True     True        27h
```

1. または、TALM マネージャーログを確認します。

   a. 以下のコマンドを実行して、TALM マネージャーの名前を取得します。

   ```
   $ oc get pod -n openshift-operators
   ```

   出力例

   ```
   NAME                                                READY   STATUS    RESTARTS   AGE
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp   2/2     Running   0          45m
   ```

   b. 以下のコマンドを実行して、TALM マネージャーログを確認します。

   ```
   $ oc logs -n openshift-operators \
   cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
   ```

   出力例

   ```
   ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error {"reconciler group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade", "name": "lab-upgrade", "namespace": "default", "error": "Cluster spoke5555 is not a ManagedCluster"} (1)
   sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
   ```

   (1) エラーメッセージには、クラスターがマネージドクラスターではないことが分かります。

マネージドクラスターが利用可能かどうかの確認

問題

ClusterGroupUpgrade CR で指定されたマネージドクラスターが利用可能かどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get managedclusters
```

出力例

```
NAME            HUB ACCEPTED   MANAGED CLUSTER URLS                  JOINED   AVAILABLE   AGE
local-cluster   true           https://api.hub.testlab.com:6443      True     Unknown     13d
spoke1          true           https://api.spoke1.testlab.com:6443   True     True        13d (1)
spoke3          true           https://api.spoke3.testlab.com:6443   True     True        27h (2)
```

(1) (2) マネージドクラスターの **AVAILABLE** フィールドの値は **True** です。

clusterSelector の確認

問題

clusterSelector フィールドが 1 つ以上のマネージドクラスターの **ClusterGroupUpgrade** CR で指定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 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:
   clusters:
   - spoke1
   - spoke3
   clusterSelector:
```

```
    - upgrade2=true
  remediationStrategy:
    canaries:
    - spoke3
    maxConcurrency: 2
    timeout: 240
```

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 **clusterSelector** ラベルに一致するクラスターのリストに存在するかどうかを確認します。

```
$ oc get managedcluster --selector=upgrade=true
```

出力例

```
NAME     HUB ACCEPTED   MANAGED CLUSTER URLS   JOINED   AVAILABLE   AGE
spoke1   true           https://api.spoke1.testlab.com:6443   True     True        13d
spoke3   true           https://api.spoke3.testlab.com:6443   True     True        27h
```

注記

クラスターは **spec.clusters** に存在し、**spec.clusterSelecter** ラベルでも一致できます。

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
```

16.8.5. 修復ストラテジー

remediationStrategy が ClusterGroupUpgrade CR に存在するかどうかの確認

問題

remediationStrategy が **ClusterGroupUpgrade** CR に存在するかどうかを確認します。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

-

出力例

```
{"maxConcurrency":2, "timeout":240}
```

ClusterGroupUpgrade CR に maxConcurrency が指定されているかどうかの確認

問題

maxConcurrency が **ClusterGroupUpgrade** CR で指定されているかどうかを確認する必要があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy.maxConcurrency}'
```

出力例

```
2
```

16.8.6. Topology Aware Lifecycle Manager

ClusterGroupUpgrade CR での条件メッセージおよびステータスの確認

問題

ClusterGroupUpgrade CR の **status.conditions** フィールドの値を確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

出力例

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"The ClusterGroupUpgrade CR has managed policies that are missing:[policyThatDoesntExist]", "reason":"UpgradeCannotStart", "status":"False", "type":"Ready"}
```

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-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy3-common-ptp-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ーのエラー

問題

TALM のマネージャーコンテナーのログを確認する必要がある場合があります。

解決方法

以下のコマンドを実行します。

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

出力例

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error {"reconciler
group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade", "name": "lab-upgrade",
"namespace": "default", "error": "Cluster spoke5555 is not a ManagedCluster"} (1)
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

(1) エラーを表示します。

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 を使用する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、PolicyGenTemplate CRD について を参照してください。

第17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。

17.1. PERFORMANCE PROFILE CREATOR の概要

Performance Profile Creator (PPC) は、Performance Addon Operator に含まれ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の作成に使用します。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

17.1.1. 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。

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 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. 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
```

2. 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>=""
```

3. **must-gather** データを保存するディレクトリーに移動します。
4. クラスターで **must-gather** を実行します。

```
$ oc adm must-gather --image=<PAO_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:v4.10 --dest-dir=must-gather
```

5. **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/
```

17.1.2. 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。

手順

1. 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
```

2. 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>
```

3. 必要に応じて、PPC ツールのヘルプを表示します。

```
$ podman run --entrypoint performance-profile-creator
registry.redhat.io/openshift4/performance-addon-rhel8-operator:v4.10 -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")
      --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
```

4. 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 /must-gather:/must-gather:z
registry.redhat.io/openshift4/performance-addon-rhel8-operator:v4.10 --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** データをコンテナー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。

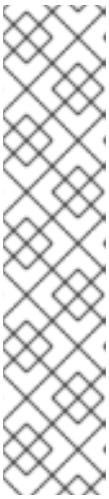
-v オプションでは、以下のいずれかへのパスを指定できます。

- **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/performance-addon-rhel8-operator:v4.10 --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=ultra-low-latency > 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:
  additionalKernelArgs:
  - nmi_watchdog=0
  - audit=0
  - mce=off
```

```
  - processor.max_cstate=1
  - intel_idle.max_cstate=0
  - idle=poll
  cpu:
    isolated: 1,3,5,7,9,11,13,15,17,19-39,41,43,45,47,49,51,53,55,57,59-79
    reserved: 0,2,4,6,8,10,12,14,16,18,40,42,44,46,48,50,52,54,56,58
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: single-numa-node
  realTimeKernel:
    enabled: true
```

7. 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、Performance Addon Operator をインストールしてください。

```
$ oc apply -f my-performance-profile.yaml
```

17.1.2.1. 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/performance-addon-rhel8-operator:v4.10 --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。

17.1.3. 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し用すると、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

- Performance Addon Operator にアクセスできる。
- **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"

PAO_IMG="registry.redhat.io/openshift4/performance-addon-rhel8-operator:v4.10"
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              Performance Addon Operator image"
  print "   -t              path to a must-gather tarball"
```

```
  ${IMG_EXISTS_CMD} "${PAO_IMG}" && ${CMD} "${PAO_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() {
  ${IMG_EXISTS_CMD} "${PAO_IMG}" || ${IMG_PULL_CMD} "${PAO_IMG}" || \
      exit_error "Performance Addon 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 "${OPT}" in
      h)
        usage
        exit 0
        ;;
      p)
        PAO_IMG="${OPTARG}"
        ;;
      t)
        MG_TARBALL="${OPTARG}"
        ;;
      ?)
        exit_error "invalid argument: ${OPTARG}"
        ;;
    esac
  done
  shift $((OPTIND - 1))

  check_requirements || exit 1
```

```
  ${CMD} -v "${DATA_DIR}:${MUST_GATHER_VOL}:z" "${PAO_IMG}" "$@" --must-gather-
dir-path "${MUST_GATHER_VOL}"
  echo "" 1>&2
}

main "$@"
```

3. 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
```

4. 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              Performance Addon 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")
         --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

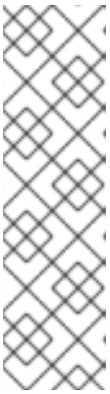
引数には、以下の 2 つのタイプがあります。

- 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: Performance Addon Operator 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/performance-addon-rhel8-operator:v4.10**。

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、Performance Addon Operator をインストールしてください。

```
$ oc apply -f my-performance-profile.yaml
```

17.1.4. Performance Profile Creator の引数

表17.1 Performance Profile Creator の引数

引数	説明
disable-ht	ハイパースレッディングを無効にします。 使用できる値は **true** または **false** です。 デフォルト: **false**。 **警告** 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。
info	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。 以下の値を使用できます。 ● **log** ● **JSON** **注記** 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。 デフォルト: **log**。
mcp-name	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。
must-gather-dir-path	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。 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。
power-consumption-mode	電力消費モード。 以下の値を使用できます。 ● **default** ● **low-latency** ● **ultra-low-latency** デフォルト: **default**。

引数	説明
profile-name	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**。
reserved-cpu-count	予約された CPU の数。このパラメーターは必須です。 **注記** 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。
rt-kernel	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。 使用できる値は **true** または **false** です。
split-reserved-cpus-across-numa	NUMA ノード全体で予約された CPU を分割します。 使用できる値は **true** または **false** です。 デフォルト: **false**。
topology-manager-policy	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。 以下の値を使用できます。 • **single-numa-node** • **best-effort** • **restricted** デフォルト: **restricted**。
user-level-networking	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。 使用できる値は **true** または **false** です。 デフォルト: **false**。

17.2. 関連情報

- **must-gather** ツールの詳細は、クラスターに関するデータの収集 を参照してください。

第18章 単一ノード OPENSHIFT でのワークロードパーティション設定

単一ノードの OpenShift デプロイメントなどのリソースに制約のある環境では、CPU リソースのほとんどを独自のワークロード用に確保し、ホスト内の固定数の CPU で実行するように OpenShift Container Platform を設定すると有利です。これらの環境では、コントロールプレーンを含む管理ワークロードは、通常のクラスターでデフォルトよりも少ないリソースを使用するように設定する必要があります。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 を分離して、予約済みの CPU セットで実行できます。

ワークロードパーティショニングを使用する場合、クラスター管理のために OpenShift Container Platform によって使用される CPU リソースは、単一ノードクラスター上のパーティション化された CPU リソースのセットに分離されます。このパーティション設定により、クラスター管理機能が定義された数の CPU に分離されます。すべてのクラスター管理機能は、その **cpuset** 設定でのみ動作します。

単一ノードクラスターの管理パーティションに必要な予約済み CPU の最低限の数は、4 つの CPU ハイパースレッド (HT) です。ベースラインの OpenShift Container Platform インストールを設定する Pod のセットと一般的なアドオン Operator のセットには、管理ワークロードパーティションに含めるためのアノテーションが付けられています。これらの Pod は、最低限のサイズの **cpuset** 設定内で正常に動作します。受け入れ可能な管理 Pod のセット以外の Operator またはワークロードを含めるには、そのパーティションに CPU HT を追加する必要があります。

ワークロードパーティション設定は、Kubernetes の通常の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、それらのコアに配置できる Pod の数を管理し、クラスター管理ワークロードとユーザーワークロードの混在を回避します。

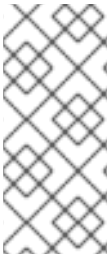
ワークロードパーティション設定を使用する場合は、Performance Addon Operator をインストールし、パフォーマンスプロファイルを適用する必要があります。

- ワークロードパーティション設定は、OpenShift Container Platform インフラストラクチャー Pod を定義済みの **cpuset** 設定に固定します。
- Performance Addon Operator のパフォーマンスプロファイルは、systemd サービスを定義済みの **cpuset** 設定に固定します。
- この **cpuset** 設定は一致する必要があります。

ワークロードパーティション設定により、定義された CPU プールまたはワークロードタイプごとに **<workload-type> .workload.openshift.io/cores** の新しい拡張リソースが導入されます。Kubelet はこれらの新しいリソースをアドバタイズし、プールに確保された Pod による CPU 要求は、通常の **cpu** リソースではなく、対応するリソース内で考慮されます。ワークロードパーティション設定が有効になっている場合、**<workload-type> .workload.openshift.io/cores** リソースにより、デフォルトの CPU プールだけでなく、ホストの CPU 容量にアクセスできます。

18.1. ワークロードの分割による CPU 割り当ての最大化

単一ノードの OpenShift クラスターのインストール中に、ワークロードの分割を有効にする必要があります。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

ワークロードパーティショニングを有効にできるのは、クラスターのインストール時のみです。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、パフォーマンスプロファイルで定義した **cpu** の値と、**MachineConfig** カスタムリソース (CR) の関連する **cpuset** の値を更新して、ワークロードパーティショニングを再設定できます。

- ワークロードの分割を有効にする base64 でエンコードされた CR には、管理ワークロードが制約される CPU セットが含まれています。**crio.conf** および **kubelet.conf** のホスト固有の値を base64 でエンコードします。この内容は、クラスターパフォーマンスプロファイルで指定されている CPU セットと一致するように調整する必要があり、クラスターホストのコア数に対して正確である必要があります。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 02-master-workload-partitioning
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8; base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMSw1Mi01MyIgf
Qo=
        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,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTEsNTItNTMiCiAgfQp
9Cg==
        mode: 420
        overwrite: true
        path: /etc/kubernetes/openshift-workload-pinning
        user:
          name: root
```

- クラスターホストで設定すると、**/etc/crio/crio.conf.d/01-workload-partitioning** の内容は次のようになります。

```
[crio.runtime.workloads.management]
activation_annotation = "target.workload.openshift.io/management"
annotation_prefix = "resources.workload.openshift.io"
```

```
[crio.runtime.workloads.management.resources]
cpushares = 0
cpuset = "0-1, 52-53" 1
```

1 **cpuset の** 値は、インストールによって異なります。

ハイパースレッディングが有効になっている場合は、各コアの両方のスレッドを指定します。**cpuset** 値は、パフォーマンスプロファイルの **spec.cpu.reserved** フィールドで定義した予約済み CPU と一致する必要があります。

- クラスターで設定すると、**/etc/kubernetes/openshift-workload-pinning** の内容は次のようになります。

```
{
  "management": {
    "cpuset": "0-1,52-53" 1
  }
}
```

1 **cpuset は、/etc/crio/crio.conf.d/01-workload-partitioning の cpuset** 値と一致する必要があります。

第19章 ネットワーク遠端のクラスター

19.1. 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。ゼロタッチプロビジョニング (ZTP) と GitOps 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。

19.1.1. 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?

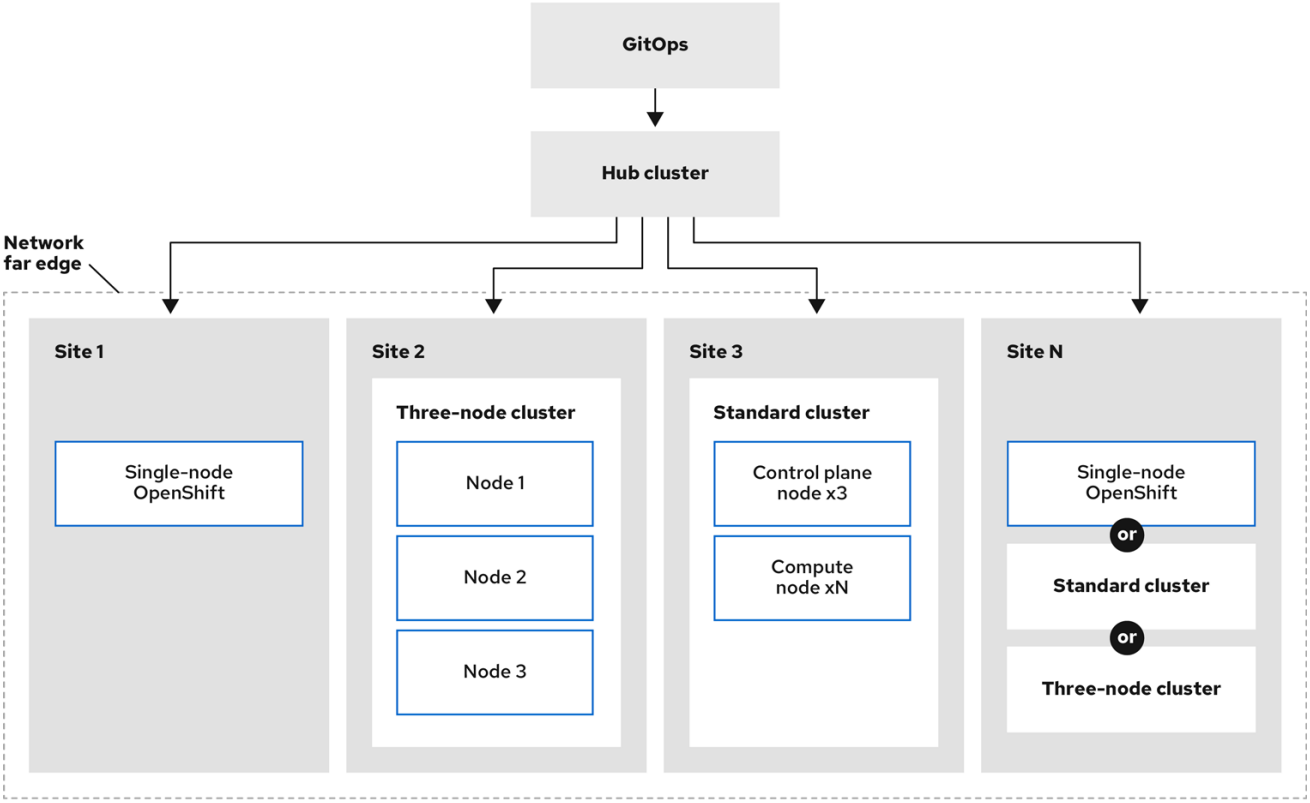
ゼロタッチプロビジョニング (ZTP) と **GitOps** は、ベアメタル機器の宣言的なサイト定義と設定を使用してリモートエッジサイトを大規模にプロビジョニングできるようにすることで、これらの課題を解決します。テンプレートまたはオーバーレイ設定は、CNF ワークロードに必要な OpenShift Container Platform 機能をインストールします。インストールとアップグレードの全ライフサイクルは、ZTP パイプラインを通じて処理されます。

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

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

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) を作成することで、ZTP ワークフローを開始します。

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



19.1.2. ZTP を使用してネットワーク遠端でクラスターをプロビジョニングする

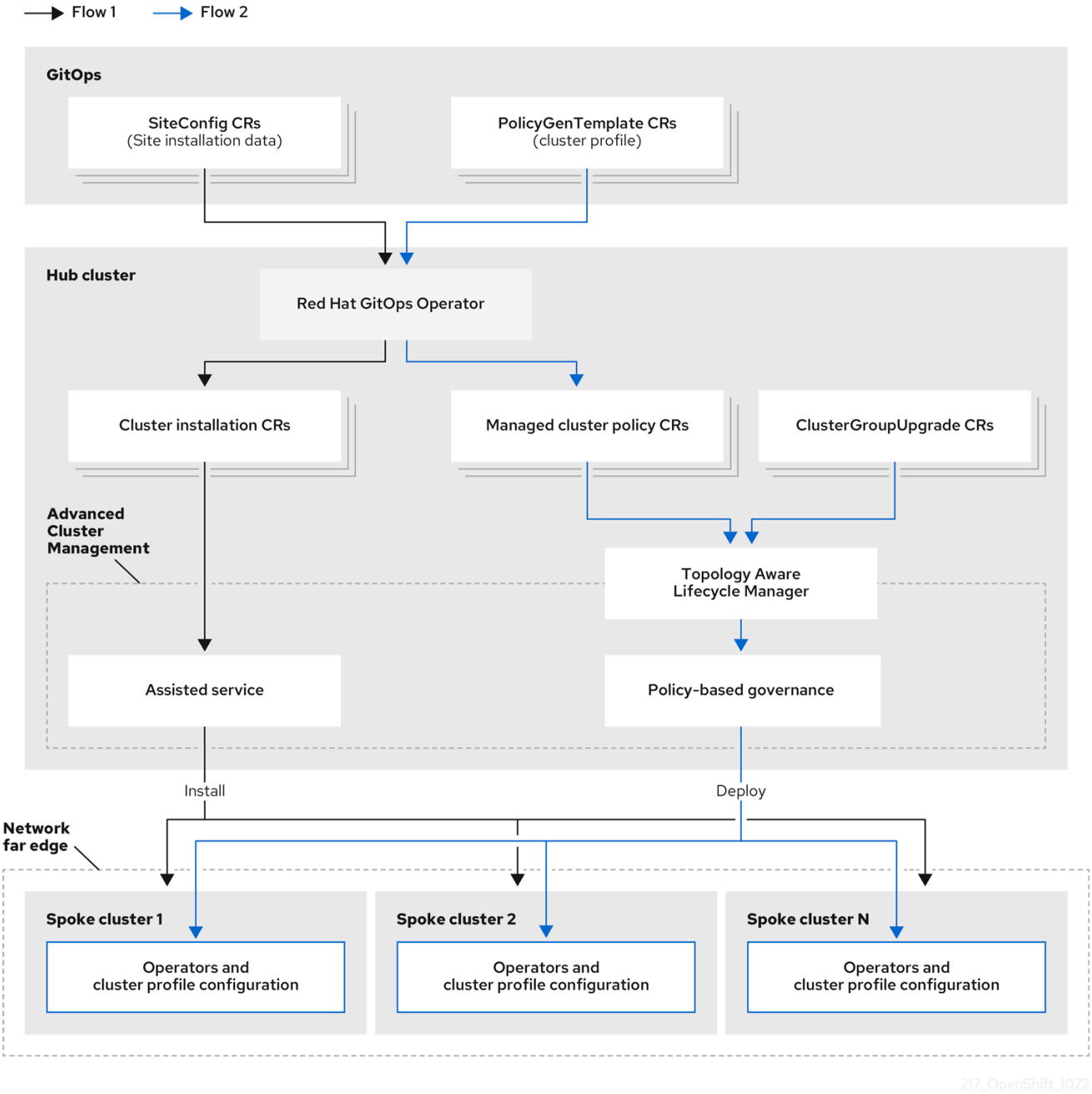
Red Hat Advanced Cluster Management (RHACM) は、単一のハブクラスターが多数のスポーククラスターを管理するハブアンドスポークアーキテクチャーでクラスターを管理します。RHACM を実行するハブクラスターは、ゼロタッチプロビジョニング (ZTP) と、RHACM のインストール時にデプロイメントされるアシストサービスを使用して、マネージドクラスターをプロビジョニングおよびデプロイメントします。

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。

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。



19.1.3. 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 ZTP は、Git リポジトリー内の **SiteConfig** カスタムリソース (CR) を使用して、OpenShift Container Platform クラスターをインストールする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 の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。

手動で、または ZTP を使用してバッチで単一のクラスターをプロビジョニングできます。

単一クラスターのプロビジョニング

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

多くのクラスターのプロビジョニング

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

19.1.4. 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

ゼロタッチプロビジョニング (ZTP) は、Red Hat Advanced Cluster Management (RHACM) を使用して、設定を適用するためのポリシーベースのガバナンスアプローチを使用してクラスターを設定します。

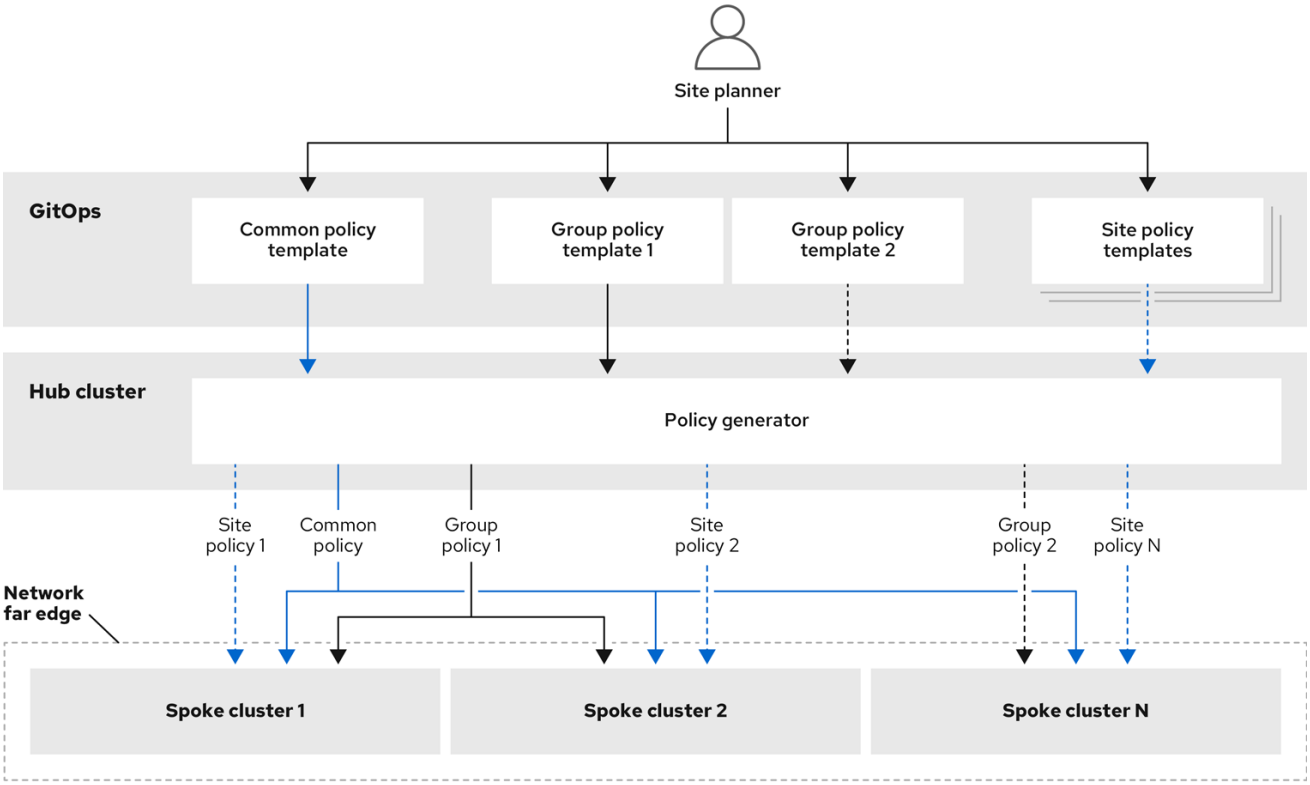
ポリシージェネレーターまたは **PolicyGen** は、簡潔なテンプレートから RHACM ポリシーを作成できるようにする GitOps Operator のプラグインです。このツールは、複数の CR を 1 つのポリシーに組み合わせることができ、フリート内のクラスターのさまざまなサブセットに適用される複数のポリシーを生成できます。

注記

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

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

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



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

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

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

表19.1 推奨される PolicyGenTemplate ポリシーカテゴリー

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

関連情報

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

19.2. ZTP 用のハブクラスターの準備

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

19.2.1. Telco RAN 4.10 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 4.10 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19.2 Telco RAN 4.10 検証済みソリューションソフトウェア

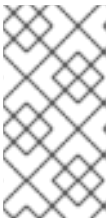
Product	ソフトウェアバージョン
Hub クラスターの OpenShift Container Platform のバージョン	4.10
GitOps ZTP プラグイン	4.9 または 4.10
Red Hat Advanced Cluster Management (RHACM)	2.4 または 2.5
Red Hat OpenShift GitOps	1.4
Topology Aware Lifecycle Manager (TALM)	4.10 (テクノロジープレビュー)

19.2.2. 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 のインストールについて参照 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

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

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

前提条件

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

重要

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

手順

1. ミラーホストにログインします。
2. mirror.openshift.com から RHCOS ISO イメージおよび RootFS イメージを取得します。以下は例になります。
   a. 必要なイメージ名と OpenShift Container Platform のバージョンを環境変数としてエクスポートします。

```
$ export ISO_IMAGE_NAME=<iso_image_name> 1
$ export ROOTFS_IMAGE_NAME=<rootfs_image_name> 1
$ export OCP_VERSION=<ocp_version> 1
```

1 ISO イメージ名 (例: **rhcos-4.10.1-x86_64-live.x86_64.iso**)

1 RootFS イメージ名 (例: **rhcos-4.10.1-x86_64-live-rootfs.x86_64.img**)

1 OpenShift Container Platform バージョン (**4.10.1** など)

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/4.10/${OCP_VERSION}/${ISO_IMAGE_NAME} -O
/var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-
v4/dependencies/rhcos/4.10/${OCP_VERSION}/${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-4.10.1-x86_64-live.x86_64.iso
rhcos-4.10.1-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

19.2.4. ハブクラスターでのアシストサービスの有効化と AgentServiceConfig の更新

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。Central Infrastructure Management (CIM) で MultiClusterHub Operator を有効にすると、アシストサービスが自動的にデプロイされます。ハブクラスターで CIM を有効にしたら、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クラスターにログインしていることを確認します。
- ハブクラスターでアシストサービスを有効にしました。詳細は、CIM の有効化 を参照してください。

手順

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

```
$ oc edit AgentServiceConfig
```

2. 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: "4.10"
  rootFSUrl: https://<host>/<path>/rhcos-live-rootfs.x86_64.img
  url: https://<mirror-registry>/<path>/rhcos-live.x86_64.iso
```

ここでは、以下のようになります。

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

<path>
ターゲットミラーレジストリー上のイメージへのパスです。

エディターを保存して終了し、変更を適用します。

19.2.5. 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) 2.4 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。

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: assisted-installer
```

```
  labels:
    app: assisted-service
data:
  ca-bundle.crt: <certificate> 1
  registries.conf: | 2
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

    [[registry]]
      location = <mirror_registry_url> 3
      insecure = false
      mirror-by-digest-only = true
```

1 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。

2 ミラーレジストリーの設定。

3 ミラーレジストリーの URL。

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
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'
  osImages:
    - openshiftVersion: <ocp_version>
      rootfs: <rootfs_url> 1
      url: <iso_url> 2
```

1 2 HTTPD サーバーの URL と一致する必要があります。

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。

19.2.6. ArgoCD を使用したハブクラスターの設定

ゼロタッチプロビジョニング (ZTP) GitOps フローに基づいて、各サイト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 ArgoCD アプリケーションのセットを使用して、ハブクラスターを設定できます。

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、ZTP GitOps プラグインコンテナーから参照デプロイメントを抽出しました。参照デプロイメントを抽出すると、次の手順で参照される **out/argocd/deployment** ディレクトリーが作成されます。

手順

1. ArgoCD パイプライン設定を準備します。
   a. example ディレクトリーと同様にディレクトリー構造で Git リポジトリーを作成します。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。
   b. ArgoCD UI を使用して、リポジトリーへのアクセスを設定します。**Settings** で以下を設定します。
      - **リポジトリー**: 接続情報を追加します。URL は **.git** などで終わって いる必要があります。**https://repo.example.com/repo.git** とクレデンシャルを指定します。
      - **certificates**: 必要に応じて、リポジトリーのパブリック証明書を追加します。
   c. 2 つの ArgoCD アプリケーション、**out/argocd/deployment/clusters-app.yaml** と **out/argocd/deployment/policies-app.yaml** を、Git リポジトリーに基づいて修正します。
      - Git リポジトリーを参照するように URL を更新します。URL は **.git** で終わります (例: **https://repo.example.com/repo.git**)。
      - **targetRevision** は、監視する Git リポジトリーブランチを示します。
      - **path** は、それぞれ **SiteConfig** CR および **PolicyGenTemplate** CR へのパスを指定します。
2. ZTP GitOps プラグインをインストールするには、以前に **out/argocd/deployment/** ディレクトリー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. 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
```

関連情報

- GitOps ZTP サイト設定リポジトリーの準備

19.2.7. GitOps ZTP サイト設定リポジトリーの準備

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

前提条件

- 必要なインストールおよびポリシーのカスタムリソース (CR) を生成するためのハブクラスター GitOps アプリケーションを設定しました。
- ゼロタッチプロビジョニング (ZTP) を使用してマネージドクラスターをデプロイしました。

手順

1. **SiteConfig** CR と **PolicyGenTemplate** CR の個別のパスを持つディレクトリー構造を作成します。
2. 以下のコマンドを使用して **ztp-site-generate** コンテナーイメージから **argocd** ディレクトリーをエクスポートします。

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.10
```

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v{product-version} 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 リポジトリー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 リポジトリー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にのみ必要です。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。

19.3. 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。ゼロタッチプライオビジョン (ZTP) パイプラインがクラスターのインストールを実行します。ZTP は、切断された環境で使用できます。

19.3.1. GitOps ZTP および Topology Aware Lifecycle Manager

GitOps ゼロタッチプロビジョニング (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。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への設定 CR の適用をオーケストレートします。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。ZTP プロセス中、OpenShift のインストール後、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 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に変更をすぐにロールアウトするリスクなしに、ポリシーを変更できます。TALM を使用して、修正されたクラスターのタイミングとセットを制御できます。

ClusterGroupUpgrade CR の自動作成

新しくデプロイされたクラスターの初期設定を自動化するために、TALM はハブクラスター上のす

べての **ManagedCluster** CR の状態を監視します。新規に作成された **ManagedCluster** CR を含む **ztp-done** ラベルを持たない **ManagedCluster** CR が適用されると、TALM は以下の特性で **ClusterGroupUpgrade** CR を自動的に作成します。

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

有効な **ClusterGroupUpgrade** の自動生成により、ユーザーの介入を必要としないゼロタッチのクラスターデプロイメントが可能になります。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、失敗した ZTP インストールを、そのクラスターの **ClusterGroupUpgrade** CR を削除するだけで再開することができます。

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ーブにインストールする必要があります。各 CR のデフォルトの波動値は、これらの前提条件を考慮したものです。

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

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

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

フェーズラベル

ClusterGroupUpgrade CR は自動的に作成され、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。インストール後の ZTP 設定開始時には、**ManagedCluster** に**ztp-running** という ラベルが貼られています。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、ZTP が適用される設定 CR のすべての調整および影響が含まれます。**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 ZTP インストール後は、このラベルを操作しないでください。

リンクされた CR

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

19.3.2. 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、ゼロタッチプロビジョニング (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。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メントするモデルを使用して、マネージドクラスターをデプロイメントします。

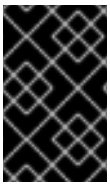
クラスターのデプロイメントには、以下が含まれます。

- 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。

19.3.3. 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。ZTP パイプラインが Baseboard Management Controller (BMC) にアクセスするためのシークレットと、アシストインストーラーサービスがレジストリーからクラスターインストールイメージを取得するためのシークレットが必要です。

注記

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

手順

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

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

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

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

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

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

4 Base64 でエンコードされたプルシークレット

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

19.3.4. SiteConfig と 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、ゼロタッチプロビジョニング (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クラスターにログインしていることを確認します。
- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセスできる必要があり、ArgoCD アプリケーションのソースリポジトリーとして設定する必要があります。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。

注記

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

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

ネットワーク接続

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

Baseboard Management Controller (BMC) の詳細

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

手順

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

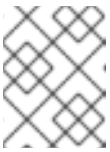
   a. 次のコマンドを実行して、クラスター namespace をエクスポートします。

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

b. namespace を作成します。

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

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



注記

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

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

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

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

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

単一ノードの OpenShift クラスター SiteConfig CR の例

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "<site_name>"
  namespace: "<site_name>"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret" 1
  clusterImageSetNameRef: "openshift-4.10" 2
  sshPublicKey: "ssh-rsa AAAA..." 3
  clusters:
  - clusterName: "<site_name>"
    networkType: "OVNKubernetes"
    clusterLabels: 4
      common: true
      group-du-sno: ""
      sites : "<site_name>"
    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" 5
  nodes:
    - hostName: "example-node.example.com" 6
      role: "master"
      #biosConfigRef:
      #  filePath: "example-hw.profile" 7
      bmcAddress: idrac-virtualmedia://<out_of_band_ip>/<system_id>/ 8
      bmcCredentialsName:
        name: "bmh-secret" 9
      bootMACAddress: "AA:BB:CC:DD:EE:11"
      bootMode: "UEFI" 10
      rootDeviceHints:
        wwn: "0x11111000000asd123"
      cpuset: "0-1,52-53"
      nodeNetwork: 11
        interfaces:
          - name: eno1
            macAddress: "AA:BB:CC:DD:EE:11"
        config:
          interfaces:
            - name: eno1
              type: ethernet
              state: up
              ipv4:
                enabled: false
              ipv6: 12
                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
```

1. **SiteConfig** CR と同じ namespace を使用して、**assisted-deployment-pull-secret** CR を作成します。

2. **clusterImageSetNameRef** は、ハブクラスターで使用可能なイメージセットを定義します。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。

3. クラスターへのアクセスに使用する SSH 公開鍵を設定します。

4 クラスターラベルは、定義した **PolicyGenTemplate** CR の **bindingRules** フィールドに対応している必要があります。たとえば、**policygentemplates/common-**

5 オプション: **KlusterletAddonConfig** で指定された CR は、クラスター用に作成されたデフォルトの **KlusterletAddonConfig** をオーバーライドするために使用されます。

6 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。

7 オプション: **biosConfigRef** を使用して、ホストに必要なファームウェアを設定します。

8 すべてのクラスタータイプに適用されます。BMC アドレスを指定します。

9 BMC 認証情報を指定する **bmh-secret** CR を作成します。**SiteConfig** CR と同じ namespace を使用します。

10 **UEFISecureBoot** を使用して、ホストでセキュアブートを有効にします。

11 ノードのネットワーク設定を指定します。

12 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。

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 リポジトリーに **sno-extra-manifest/** などのディレクトリーを作成し、このディレクトリーにカスタムマニフェストの CR を追加します。**SiteConfig.yaml** が **extraManifestPath** フィールドでこのディレクトリーを参照する場合、この参照ディレクトリーの CR はすべて、デフォルトの追加マニフェストセットに追加されます。

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

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

関連情報

- GitOps ZTP サイト設定リポジトリーの準備
- ArgoCD を使用したハブクラスターの設定

- バリデーターインフォームポリシーを使用した 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

19.3.5. 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クラスターにログインしていることを確認します。

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]'
   ```

19.3.6. 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クラスターにログインしていることを確認します。

手順

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

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

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

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

```
$ oc get managedcluster
```

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

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

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

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

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

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

19.3.7. ZTP パイプラインからマネージドクラスターサイトを削除

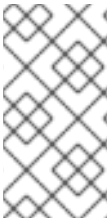
ZTP パイプラインから、マネージドサイトと、関連するインストールおよび設定ポリシーの CR を削除できます。

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クラスターにログインしていることを確認します。

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。
   ZTP パイプラインを再度実行すると、生成された CR が削除されます。
2. 任意: サイトを永続的に削除する場合は、Git リポジトリーから **SiteConfig** ファイルおよびサイト固有の **PolicyGenTemplate** ファイルも削除する必要があります。
3. 任意: たとえば、サイトを再デプロイする際にサイトを一時的に削除する場合には、Git リポジトリーに **SiteConfig** およびサイト固有の **PolicyGenTemplate** CR を残しておくことができます。



注記

Git リポジトリーから **SiteConfig** ファイルを削除した後、対応するクラスターがデタッチプロセスで停止する場合は、デタッチされたクラスターのクリーンアップに関する情報について、ハブクラスターの Red Hat Advanced Cluster Management (RHACM) を確認してください。

関連情報

- クラスターの削除は、管理からクラスターを削除する を参照してください。

19.3.8. 古いコンテンツを ZTP パイプラインから削除する

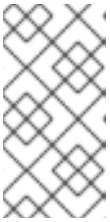
ポリシーの名前を変更した場合など、**PolicyGenTemplate** 設定を変更した結果、古いポリシーが作成された場合は、次の手順を使用して古いポリシーを削除します。

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クラスターにログインしていることを確認します。

手順

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



注記

Git リポジトリーからゼロタッチプロビジョニング (ZTP) ポリシーを削除し、その結果、ハブクラスターからもポリシーを削除しても、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>
```

19.3.9. ZTP パイプラインの解体

ArgoCD パイプラインと生成されたすべての 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クラスターにログインしていることを確認します。

手順

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

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

3. 変更をコミットして、サイトリポジトリーにプッシュします。

19.4. 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。

19.4.1. 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** 参照コンテナーから抽出された **PolicyGenTemplate** CR (**common-du-ranGen.yaml**) を示しています。**common-du-ranGen.yaml** ファイルは、2 つの Red Hat Advanced Cluster Management (RHACM) ポリシーを定義します。ポリシーは、CR 内の **policyName** の一意の値

ごとに 1 つずつ、設定 CR のコレクションを管理します。**common-du-ranGen.yaml** は、単一の配置バインディングと配置ルールを作成して、**bindingRules** セクションにリストされているラベルに基づいてポリシーをクラスターにバインドします。

PolicyGenTemplate CR の例 - common-du-ranGen.yaml

```
---
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "common"
  namespace: "ztp-common"
spec:
  bindingRules:
    common: "true" (1)
  sourceFiles: (2)
    - fileName: SriovSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: SriovOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionNS.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpSubscriptionOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: PtpOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogNS.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: ClusterLogOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageNS.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperGroup.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageSubscription.yaml
      policyName: "subscriptions-policy"
    - fileName: StorageOperatorStatus.yaml
      policyName: "subscriptions-policy"
    - fileName: ReduceMonitoringFootprint.yaml
      policyName: "config-policy"
    - fileName: OperatorHub.yaml (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-ptp-slave"
      spec:
        profile:
        - name: "slave"
          interface: "ens5f0"
          ptp4lOpts: "-2 -s --summary_interval -4"
          phc2sysOpts: "-a -r -n 24"
```

ソースファイル **PtpConfigSlave.yaml** を例として使用すると、ファイルは **PtpConfig** CR を定義します。**PtpConfigSlave** サンプルの生成ポリシーは **group-du-sno-config-policy** という名前です。生成された **group-du-sno-config-policy** に定義される **PtpConfig** CR は **du-ptp-slave** という名前です。**PtpConfigSlave.yaml** で定義された **spec** は、**du-ptp-slave** の下に、ソースファイルで定義された他の **spec** 項目と共に配置されます。

次の例は、**group-du-sno-config-policy** CR を示しています。

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: group-du-ptp-config-policy
  namespace: groups-sub
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/standards: NIST SP 800-53
spec:
    remediationAction: inform
    disabled: false
    policy-templates:
        - objectDefinition:
            apiVersion: policy.open-cluster-management.io/v1
            kind: ConfigurationPolicy
            metadata:
                name: group-du-ptp-config-policy-config
            spec:
                remediationAction: inform
                severity: low
                namespaceselector:
                    exclude:
                        - kube-*
                    include:
                        - '*'
                object-templates:
                    - complianceType: musthave
                      objectDefinition:
                        apiVersion: ptp.openshift.io/v1
                        kind: PtpConfig
                        metadata:
                            name: du-ptp-slave
                            namespace: openshift-ptp
                        spec:
                            recommend:
                                - match:
                                - nodeLabel: node-role.kubernetes.io/worker-du
                                  priority: 4
                                  profile: slave
                            profile:
                                - interface: ens5f0
                                  name: slave
                                  phc2sysOpts: -a -r -n 24
                                  ptp4lConf: |
                                    [global]
                                    #
                                    # Default Data Set
                                    #
                                    twoStepFlag 1
                                    slaveOnly 0
                                    priority1 128
```

```
priority2 128
domainNumber 24
.....
```

19.4.2. 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。

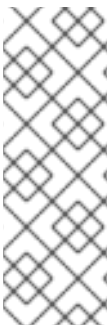
- 切断された環境では、すべての Operator を含む単一のインデックスとしてレジストリーを設定することにより、すべての Operator に対して単一のカタログソースを使用します。マネージドクラスターに **CatalogSource** CR を追加するたびに、CPU 使用率が増加します。

- **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リングに関する推奨事項は、パフォーマンスおよびスケーラビリティー を参照してください。



注記

ハブクラスターで多数のスポーククラスターを管理する場合は、ポリシーの数を最小限に抑えてリソースの消費を減らします。

複数のコンフィギュレーション CR を 1 つまたは限られた数のポリシーにグループ化することは、ハブクラスター上のポリシーの総数を減らすための 1 つの方法です。サイト設定の管理に共通、グループ、サイトというポリシーの階層を使用する場合は、サイト固有の設定を 1 つのポリシーにまとめることが特に重要である。

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

GitOps ZTP コンテナーから取得した参照設定は、RAN (Radio Access Network) 分散ユニット (DU) アプリケーションに典型的な厳しいパフォーマンスとリソース利用制約をクラスターが確実にサポートできるように、重要な機能とノードのチューニング設定のセットを提供するように設計されています。

ベースライン設定の変更または省略は、機能の可用性、パフォーマンス、およびリソースの利用に影響を与える可能性があります。参照 **PolicyGenTemplate** CR をベースに、お客様のサイト要件に合わせた設定ファイルの階層を作成します。

RAN DU クラスター設定に定義されているベースライン **PolicyGenTemplate** CR は、GitOps ZTP **ztp-site-generate** コンテナーから抽出することが可能です。詳細は、「GitOps ZTP サイト設定リポジトリーの準備」を参照してください。

PolicyGenTemplate の CR は、**./out/argocd/example/policygentemplates** フォルダーに格納されています。参照アーキテクチャーには、common、group、および site 固有の設定 CR があります。各 **PolicyGenTemplate** CR は **./out/source-crs** フォルダーにある他の CR を参照します。

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

表19.3 RAN デプロイメントの PolicyGenTemplate CR

PolicyGenTemplate CR	説明
example-multinode-site.yaml	マルチノードクラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
example-sno-site.yaml	単一ノードの OpenShift クラスターに適用される一連の CR が含まれています。これらの CR は、RAN インストールに典型的な SR-IOV 機能を設定します。
common-ranGen.yaml	すべてのクラスターに適用される共通の RAN CR のセットが含まれています。これらの CR は、RAN の典型的なクラスター機能とベースラインクラスターのチューニングを提供する Operator のセットをサブスクライブします。
group-du-3node-ranGen.yaml	3 ノードクラスター用の RAN ポリシーのみが含まれています。
group-du-sno-ranGen.yaml	シングルノードクラスター用の RAN ポリシーのみが含まれています。
group-du-standard-ranGen.yaml	標準的な 3 つのコントロールプレーンクラスターの RAN ポリシーが含まれています。
group-du-3node-validator-ranGen.yaml	**PolicyGenTemplate** CR は、3 ノードクラスターに必要なさまざまなポリシーを生成するために使用されます。
group-du-standard-validator-ranGen.yaml	標準クラスターに必要なさまざまなポリシーを生成するために使用される **PolicyGenTemplate** CR。

PolicyGenTemplate CR	説明
group-du-sno-validator-ranGen.yaml	**PolicyGenTemplate** CR は、単一ノードの OpenShift クラスターに必要なさまざまなポリシーを生成するために使用されます。

関連情報

- GitOps ZTP サイト設定リポジトリーの準備

19.4.4. 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、ゼロタッチプロビジョニング (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クラスターにログインしていることを確認します。
- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. サイト固有の設定 CR の **PolicyGenTemplate** CR を作成します。
   a. CR の適切な例を **out/argocd/example/policygentemplates** フォルダーから選択します (**example-sno-site.yaml** または **example-multinode-site.yaml**)。
   b. サンプルファイルの **bindingRules** フィールドを、**SiteConfig** CR に含まれるサイト固有のラベルと一致するように変更します。サンプルの **SiteConfig** ファイルでは、サイト固有のラベルは **sites: example-sno** です。

      注記

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

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

3. オプション: フリート内のクラスターの特定のグループに適用されるグループ設定 CR の **PolicyGenTemplate** CR を作成します。
オーバーレイド仕様ファイルの内容が必要な終了状態と一致することを確認します。
out/source-crs ディレクトリーには、PolicyGenTemplate テンプレートに含めることができる source-crs の完全な一覧が含まれます。

注記

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

   a. **out/argocd/example/policygentemplates** フォルダーから CR の適切な例を選択します (例: **group-du-sno-ranGen.yaml**)。

   b. サンプルファイルの内容を目的の設定に合わせて変更します。

4. オプション:ZTP のインストールとデプロイされたクラスターの設定が完了したときに通知するバリデータ通知ポリシー **PolicyGenTemplate** CR を作成します。詳細は、バリデータ通知ポリシーの作成を参照してください。

5. **out/argocd/example/policygentemplates/ns.yaml** ファイルの例と同様の YAML ファイルで、すべてのポリシーの namespace を定義してください。

重要

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

6. **out/argocd/example/policygentemplates/kustomization.yaml** に示されている例と同様に、**PolicyGenTemplate** CR と **Namespace** CR をジェネレーターセクションの **kustomization.yaml** ファイルに追加します。

7. **PolicyGenTemplate** CR、**Namespace** CR、および関連する **kustomization.yaml** ファイルを Git リポジトリーにコミットし、変更をプッシュします。
ArgoCD パイプラインが変更を検出し、マネージドクラスターのデプロイを開始します。**SiteConfig** CR と **PolicyGenTemplate** CR に同時に変更をプッシュすることができます。

関連情報

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

19.4.5. 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クラスターにログインしていることを確認します。

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": "The ClusterGroupUpgrade CR has upgrade policies that are still non
   compliant",
     "reason": "UpgradeNotCompleted",
     "status": "False",
     "type": "Ready"
   }
   ```

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 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。

19.4.6. 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-wraped CR のセットが表示されるはずです。

ポリシーの同期に失敗した場合は、以下のトラブルシューティング手順を使用します。

手順

1. ポリシーの詳細情報を表示するには、次のコマンドを実行します。

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

2. **Status: Conditions:** の有無を確認し、エラーログを表示します。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に注意してください。

```
$ 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。

19.4.7. 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、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。

19.4.8. ZTP インストールの実行内容の表示

Zero touch provisioning (ZTP) は、クラスターの ZTP インストール状況を確認するプロセスを簡素化します。ZTP の状態は、クラスターのインストール、クラスターの設定、ZTP の完了の 3 つのフェーズで推移します。

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。

ZTP が完了

クラスターのインストールと設定は、ZTP の実行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 done 状態への遷移は、Red Hat Advanced Cluster Management (RHACM) のバリデータ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 ZTP プロビジョニングが完了したときにのみ、準拠した状態に移行することを検証するものです。

バリデータ通知ポリシーは、クラスターの設定が完全に適用され、Operator が初期化を完了したことを確認します。ポリシーは以下を検証します。

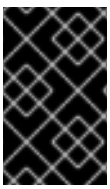
- ターゲット **MachineConfigPool** には予想されるエントリーが含まれ、更新が完了しました。全ノードが利用可能で、低下することはありません。
- SR-IOV Operator は、**syncStatus: Succeeded** の 1 つ以上の **SriovNetworkNodeState** によって示されているように初期化を完了しています。
- PTP Operator デーモンセットが存在する。

19.5. 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

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。

19.5.1. ZTP のインストールと設定の CR を手動で生成する

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていることを確認します。

手順

1. 次のコマンドを実行して、出力フォルダーを作成します。

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

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.10 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
        │   ├── kustomization.yaml
        │   └── ns.yaml
        └── siteconfig
            ├── example-sno.yaml
            ├── KlusterletAddonConfigOverride.yaml
            └── kustomization.yaml
```

3. サイトインストール CR の出力フォルダーを作成します。

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

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

単一ノードの OpenShift クラスター SiteConfig CR の例

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

```
pullSecretRef:
  name: "assisted-deployment-pull-secret" 1
clusterImageSetNameRef: "openshift-4.10" 2
sshPublicKey: "ssh-rsa AAAA..." 3
clusters:
- clusterName: "<site_name>"
  networkType: "OVNKubernetes"
  clusterLabels: 4
    common: true
    group-du-sno: ""
    sites : "<site_name>"
  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" 5
  nodes:
    - hostName: "example-node.example.com" 6
      role: "master"
      #biosConfigRef:
      #  filePath: "example-hw.profile" 7
      bmcAddress: idrac-virtualmedia://<out_of_band_ip>/<system_id>/ 8
      bmcCredentialsName:
        name: "bmh-secret" 9
      bootMACAddress: "AA:BB:CC:DD:EE:11"
      bootMode: "UEFI" 10
      rootDeviceHints:
        wwn: "0x11111000000asd123"
      cpuset: "0-1,52-53"
      nodeNetwork: 11
        interfaces:
          - name: eno1
            macAddress: "AA:BB:CC:DD:EE:11"
        config:
          interfaces:
            - name: eno1
              type: ethernet
              state: up
              ipv4:
                enabled: false
              ipv6: 12
                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
```

1. **SiteConfig** CR と同じ namespace を使用して、 **assisted-deployment-pull-secret** CR を作成します。
2. **clusterImageSetNameRef** は、ハブクラスターで使用可能なイメージセットを定義します。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。
3. クラスターへのアクセスに使用する SSH 公開鍵を設定します。
4. クラスターラベルは、定義した **PolicyGenTemplate** CR の **bindingRules** フィールドに対応している必要があります。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。
5. オプション: **KlusterletAddonConfig** で指定された CR は、クラスター用に作成されたデフォルトの **KlusterletAddonConfig** をオーバーライドするために使用されます。
6. 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。
7. オプション: **biosConfigRef** を使用して、ホストに必要なファームウェアを設定します。
8. すべてのクラスタータイプに適用されます。BMC アドレスを指定します。
9. BMC 認証情報を指定する **bmh-secret** CR を作成します。**SiteConfig** CR と同じ namespace を使用します。
10. **UEFISecureBoot** を使用して、ホストでセキュアブートを有効にします。
11. ノードのネットワーク設定を指定します。
12. 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。

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:v4.10.1 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:v4.10.1 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:v4.10.1 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
```

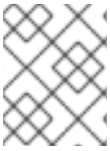
8. クラスターのインストールに使用する CR のベースとして、生成された CR を使用します。「単一のマネージドクラスターのインストール」で説明されているように、インストール CR をハブクラスターに適用します。設定 CR は、クラスターのインストールが完了した後にクラスターに適用できます。

関連情報

- 単一のマネージドクラスターのインストール
- BMC アドレス指定

19.5.2. 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。ZTP パイプラインが Baseboard Management Controller (BMC) にアクセスするためのシークレットと、アシストインストーラーサービスがレジストリーからクラスターインストールイメージを取得するためのシークレットが必要です。



注記

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

手順

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

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

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
```

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

1. 関連する **SiteConfig** CR で設定された namespace と一致する必要があります
2. **password** と **username** の Base64 エンコード値
3. 関連する **SiteConfig** CR で設定された namespace と一致する必要があります
4. Base64 でエンコードされたプルシークレット

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

19.5.3. 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クラスターにログインしていることを確認します。
- 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-4.10.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-4.10.0-rc.0 1
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.10.0-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

2. **clusterImageSet** CR を適用します。

```
$ oc apply -f clusterImageSet-4.10.yaml
```

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

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

1 2 プロビジョニングするマネージドクラスターの名前。

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

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

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

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

関連情報

- マネージドクラスターネットワークの接続の前提条件

19.5.4. マネージドクラスターのインストールステータスの監視

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

前提条件

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

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

True はマネージドクラスターの準備が整っていることを示します。

2. エージェントのステータスを確認します。

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

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

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

4. クラスターのプロビジョニングのステータスを確認します。

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

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

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

6. マネージドクラスターのアドオンサービスのステータスを確認します。

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

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

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

19.5.5. マネージドクラスターのトラブルシューティング

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

手順

1. マネージドクラスターのステータスを確認します。

```
$ oc get managedcluster
```

出力例

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

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

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

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

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

出力例

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

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

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

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

4. エラーを解決し、クラスターをリセットします。

   a. クラスターのマネージドクラスターリソースを削除します。

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

   b. クラスターの namespace を削除します。

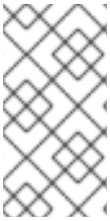
```
$ oc delete namespace <cluster_name>
```

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

   c. マネージドクラスターのカスタムリソースを再作成します。

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

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



注記

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

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

表19.4 RHACM によって生成されたクラスターインストール CR

CR	説明	使用法
BareMetalHost	ターゲットのベアメタルホストの Baseboard Management Controller (BMC) の接続情報が含まれています。	Redfish プロトコルを使用して、ターゲットサーバーで検出イメージをロードおよび起動するために BMC にアクセスできます。
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。これは、DHCP が使用される場合は必要ありません。	マネージドクラスターの Kube API サーバーの静的 IP アドレスを設定します。
Agent	ターゲットのベアメタルホストに関するハードウェア情報が含まれています。	ターゲットマシンの検出イメージの起動時にハブ上に自動的に作成されます。
ManagedCluster	クラスターがハブで管理されている場合は、インポートして知られている必要があります。この Kubernetes オブジェクトはそのインターフェイスを提供します。	ハブは、このリソースを使用してマネージドクラスターのステータスを管理し、表示します。
KlusterletAddonConfig	**ManagedCluster** リソースにデプロイされるハブによって提供されるサービスのリストが含まれます。	**ManagedCluster** リソースにデプロイするアドオンサービスをハブに指示します。
Namespace	ハブ上にある **ManagedCluster** リソースの論理領域。サイトごとに一意です。	リソースを **ManagedCluster** に伝搬します。
Secret	**BMC Secret** と **Image Pull Secret** の 2 つの CR が作成されます。	• **BMC Secret** は、ユーザー名とパスワードを使用して、ターゲットのベアメタルホストに対して認証を行います。 • **Image Pull Secre** には、ターゲットベアメタルホストにインストールされている OpenShift Container Platform イメージの認証情報が含まれます。

CR	説明	使用法
ClusterImageSet	リポジトリーおよびイメージ名などの OpenShift Container Platform イメージ情報が含まれます。	OpenShift Container Platform イメージを提供するためにリソースに渡されます。

19.6. 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 OPENSHIFT クラスター設定

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

関連情報

- 単一クラスターを手動でデプロイするには、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 ゼロタッチプロビジョニング (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、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。

19.6.1. OpenShift Container Platform で低レイテンシーのアプリケーションを実行する

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

RHCOS のリアルタイムカーネル

ワークロードが高レベルのプロセス決定で処理されるようにします。

CPU の分離

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

NUMA 対応のトポロジー管理

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

Huge Page のメモリー管理

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

PTP を使用した精度同期

サブマイクロ秒の正確性を持つネットワーク内のノード間の同期を可能にします。

19.6.2. 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19.5 最小リソース要件

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

注記

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

- (コアあたりのスレッド数×コア)×ソケット= vCPU

重要

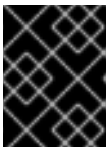
仮想メディアを使用して起動する場合は、サーバーには Baseboard Management Controller (BMC) が必要です。

19.6.3. 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19.6 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

ファームウェア設定	設定
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。

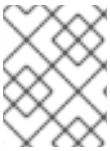
19.6.4. マネージドクラスターネットワークの接続の前提条件

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

19.6.5. 推奨されるインストール時のクラスター設定

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。



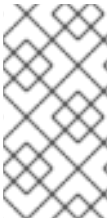
注記

クラスターデプロイメントに ZTP GitOps 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。

19.6.5.1. 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、パフォーマンスプロファイルおよび関連する **MachineConfig** カスタムリソース (CR) で定義した **cpu** 値を更新することで、ワークロードの分割を再設定できます。

- ワークロードの分割を有効にする base64 でエンコードされた CR には、管理ワークロードが制約される CPU セットが含まれています。**crio.conf** および **kubelet.conf** のホスト固有の値を base64 でエンコードします。クラスターパフォーマンスプロファイルで指定されている CPU セットに一致するように内容を調整します。クラスターホストのコア数と一致する必要があります。

  推奨されるワークロードパーティショニング設定

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 02-master-workload-partitioning
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8; base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlvbl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVu
dCIKYW5ub3RhdGlvbl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbyIKcmVzb3VyY2VzID0geyAiY3B1c2hhcmVzIiA9IDAsICJjcHVzZXQiID0gIjAtMSw1Mi01MyIgf
Qo=
        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,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6ICIwLTEsNTItNTMiCiAgfQp
9Cg==
      mode: 420
      overwrite: true
      path: /etc/kubernetes/openshift-workload-pinning
      user:
        name: root
```

- クラスターホストで設定すると、**/etc/crio/crio.conf.d/01-workload-partitioning** の内容は次のようになります。

```
[crio.runtime.workloads.management]
activation_annotation = "target.workload.openshift.io/management"
annotation_prefix = "resources.workload.openshift.io"
resources = { "cpushares" = 0, "cpuset" = "0-1,52-53" } (1)
```

(1) **CPUs** の値は、インストールによって異なります。

ハイパースレッディングが有効になっている場合は、各コアの両方のスレッドを指定します。**CPUs** の値は、パフォーマンスプロファイルで指定された予約済み CPU セットと一致する必要があります。

- クラスターで設定すると、**/etc/kubernetes/openshift-workload-pinning** の内容は次のようになります。

```
{
  "management": {
    "cpuset": "0-1,52-53" (1)
  }
}
```

(1) **cpuset** は、**/etc/crio/crio.conf.d/01-workload-partitioning** の **CPUs** 値と一致する必要があります。

19.6.5.2. プラットフォーム管理フットプリントの削減

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

推奨されるコンテナーマウント namespace の設定

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: container-mount-namespace-and-kubelet-conf-master
spec:
  config:
    ignition:
```

```
    version: 3.2.0
  storage:
    files:
    - contents:
        source: data:text/plain;charset=utf-
8; base64,IyEvYmluL2Jhc2gKCmRlYnVnKCkgewogIGVjaG8gJEAgPiYyCn0KCnVzYWdlKCkgewogIGVj
aG8gVXNhZ2U6ICQoYmFzZW5hbWUgJDApIFVOSVQgW2VudmZpbGUgW3Zhcm5hbWVdXQogIGV
jaG8KICBlY2hvIEV4dHJhY3QgdGhlIGNvbnRlbnRzIG9mIHRoZSBmaXJzdCBFeGVjU3RhcnQgc3Rhbn
phIGZyb20gdGhlIGdpdmVuIHN5c3RlbWQgdW5pdCBhbmQgcmV0dXJuIGl0IHRvIHN0ZG91dAogIGVj
aG8KICBlY2hvICJJZiAnZW52ZmlsZScgaXMgcHJvdmlkZWQsIHB1dCBpdCBpbiB0aGVyZSBpbnN0ZW
FkLCBhcyBhbiBlbnZpcm9ubWVudCB2YXJpYWJsZSBuYW1lZCAndmFybmFtZSciCiAgZWNobyAiRGV
mYXVsdCAndmFybmFtZScgaXMgRVhFQ1NUQVJUIGlmIG5vdCBzcGVjaWZpZWQiCiAgZXhpdCAxC
n0KCIVOSVQ9JDEKRU5WRklMRT0kMgpWQVJOQU1FPSQzCmlmIFtbIC16ICRVTklUIHx8ICRVTklUI
D09ICItLWhlbHAiIHx8ICRVTklUID09ICItaCIgXV07IHRoZW4KICB1c2FnZQpmaQpkZWJ1ZyAiRXh0cm
FjdGluZyBFeGVjU3RhcnQgZnJvbSAkVU5JVCIKRklMRT0kKHN5c3RlbWN0bCBjYXQgJFVOSVQgfCB
oZWFkIC1uIDEpCkZJTEU9JHtGSUxFI1wjIH0KaWYgW1sgISAtZiAkRklMRSBdXTsgdGhlbgogIGRlYnV
nICJGYWlsZWQgdG8gZmluZCByb290IGZpbGUgZm9yIHVuaXQgJFVOSVQgKCRGSUxFKSIKICBle
Gl0CmZpCmRlYnVnICJTZXJ2aWNlIGRlZmluaXRpb24gaXMgaW4gJEZJTEUiCkVYRUNTVEFSVD0k
KHNlZCAtbiAtZSAnL15FeGVjU3RhcnQ9LipcXCQvIHsgcy9eRXhlY1N0YXJ0PS8vOyBw
IH0nIC1lICcvXkV4ZWNTdGFydD0uKlteXFxdJC8geyBzL15FeGVjU3RhcnQ9Ly87IHAgfSBcCgppZiBbWyAkRU5WRklMRSBdXTsgdGhlbgogIFZBUk5BTUU9JHtWQVJOQU1FOi1FWEVDU1RBUlR9CiAgZWNobyAiJHtWQVJOQU1FfT0ke0VYRUNTVEFSVH0iID4gJEVOVkZJTEUKZWxzZQogIGVja
G8gJEVYRUNTVEFSVApmaQo=
      mode: 493
      path: /usr/local/bin/extractExecStart
    - contents:
        source: data:text/plain;charset=utf-
8; base64,IyEvYmluL2Jhc2gKbnNlbnRlciAtLW1vdW50PS9ydW4vY29udGFpbmVyLW1vdW50LW5hbWV
zcGFjZS9tbnQgIiRAIgo=
      mode: 493
      path: /usr/local/bin/nsenterCmns
  systemd:
    units:
    - contents: |
        [Unit]
        Description=Manages a mount namespace that both kubelet and crio can use to share their
container-specific mounts

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

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

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

19.6.5.3. 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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module
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
```

19.6.5.4. コンテナーの起動の高速化

次の **MachineConfig** CR は、コア OpenShift プロセスとコンテナーを設定して、システムの起動とシャットダウン中に利用可能なすべての CPU コアを使用します。これにより、初回起動および再起動中のシステムリカバリーが加速されます。

推奨される高速化されたコンテナーの起動設定

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 04-accelerated-container-startup-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8; base64,IyEvYmluL2Jhc2gKIwojIFRlbXBvcmFyaWx5IHJlc2V0IHRoZSBjb3JlIHN5c3RlbSBwcm9jZXNz
ZXMncyBDUFUgYWZmaW5pdHkgdG8gYmUgdW5yZXN0cmljdGVkIHRvIGFjY2VsZXJhdGUgc3Rhcn
R1cCBhbmQgc2h1dGRvd24KIwojIFRoZSBkZWZhdWx0cyBiZWxvdyBjYW4gYmUgb3ZlcnJpZGRlbiB2a
WEgZW52aXJvbm1lbnQgdmFyaWFibGVzCiMKCiMgVGhlIGRlZmF1bHQgc2V0IG9mIGNyaXRpY2FsI
HByb2Nlc3NlcyB3aG9zZSBhZmZpbml0eSBzaG91bGQgYmUgdGVtcG9yYXJpbHkgdW5ib3VuZDoKQ
1JJVElDQUxfUFJPQ0VTU0VTPSR7Q1JJVElDQUxfUFJPQ0VTU0VTOi0ic3lzdGVtZCBvdnMgY3JpbyB
rdWJlbGV0IE5ldHdvcmtNYW5hZ2VyIGNvbm1vbiBkYnVzIn0KCiMgRGVmYXVsdCB3YWl0IHRpbWUga
XMgNjAwcyA9IDEwbToKTUFYSU1VTV9XQUlUX1RJTUU9JHtNQVhJTVVNX1dBSVRfVElNRTotNjAw
fQoKIyBEZWZhdWx0IHN0ZWFkeS1zdGF0ZSB0aHJlc2hvbGQgPSAyJQojIEFsbG93ZWQgdmFsdWV
zOgojICA0ICAtIGFic29sdXRlIHBvZCBjb3VudCAoKy8tKQojICA0JSAtIHBlcmNlbnQgY2hhbmdlICgrLy0p
CiMgIC0xIC0gZGlzYWJsZSB0aGUgc3RlYWR5LXN0YXRlIGNoZWNrCINURUFEWV9TVEFURV9USF
JFU0hPTEQ9JHtTVEVBRFlfU1RBVEVfTEVhSRVNIT0xEOi0yJX0KCiMgRGVmYXVsdCBzdGVhZHktc3
RhdGUgd2luZG93ID0gNjBzCiMgSWYgdGhlIHJ1bm5pbmcgcG9kIGNvdW50IHN0YXlzIHdpdGhpbiB0a
GUgZ2l2ZW4gdGhyZXNob2xkIGZvciB0aGlzIHRpbWUKIyBwZXJpb2QsIHJldHVybiBDUFUgdXRpbGl6
YXRpb24gdG8gbm9ybWFsIGJlZm9yZSB0aGUgbWF4aW11bSB3YWl0IHRpbWUgaGFzCiMgZXhwaX
JlcwpTVEVBRFlfU1RBVEVfV0lORE9XPSR7U1RFQURZX1NUQVRFX1dJTkRPVzotNjB9CgojIERlZm
F1bHQgc3RlYWR5LXN0YXRlIGFsbG93cyBhbnkgcG9kIGNvdW50IHRvIGJlICJzdGVhZHkgc3RhdGUi
CiMgSW5jcmVhc2luZyB0aGlzIHdpbGwgc2tpCBhbnkgc3RlYWR5LXN0YXRlIGNoZWNrcyB1bnRpbCB
0aGUgY291bnQgcmlzZXMgYWJvdmUKIyB0aGlzIG51bWJlciB0byBhdm9pZCBmYWxzZSBwb3NpdGl2
ZXMgaWYgdGhlcmUgYXJlIHNvbWUgcGVyaW9kcyB3aGVyZSB0aGUKIyBjb3VudCBkb2Vzbid0IGluY3
JlYXNlIGJ1dCB3ZSBrbm93IHdlIGNhbid0IGF0IHN0ZWFkeS1zdGF0ZSB5ZXQuCINURUFEWV9T
VEFURV9NSU5JTVVNPSR7U1RFQURZX1NUQVRFX01JTklNVU06LTB9CgojIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjCgpLVUJFTEVUX0NQVV9TVEFURT0vdmFyL2xpY
9rdWJlbGV0L2NwdV9tYW5hZ2VyX3N0YXRlCkZVTExfQ1BVX1NUQVRFPS9zeXMvZnMvY2dyb3VwL
2NwdXNldC9jcHVzZXQuY3B1cwp1bnJlc3RyaWN0ZWRDcHVzZXQoKSB7CiAgbG9jYWwgY3B1cwogI
GlmIFtbIC1lICRLVUJFTEVUX0NQVV9TVEFURSBdXTsgdGhlbgogICAgICBjcHVzPSQoanEgLXIgJy5k
ZWZhdWx0Q3B1U2V0JyA8JEtVQkVMRVRfQ1BVX1NUQVRFKQogICAgICBpZiAgaWYgW1sgLXogJGNw
```

dXMgXV07IHRoZW4KICAgICMgZmFsbCBiYWNrIHRvIHVzaW5nIGFsbCBjcHVzIGlmIHRoZSBrdWJlb
GV0IHN0YXRlIGlzIG5vdCBjb25maWd1cmVkIHlldAogICAgW1sgLWUgJEZVTExfQ1BVX1NUQVRFIF1
dIHx8IHJldHVybiAxCiAgICBjcHVzPSQoPCRGVUxMX0NQVV9TVEFURSkKICBmaQogIGVjaG8gJGN
wdXMKfQoKcmVzdHJpY3RlZENwdXNldCgpIHsKICBmb3IgYXJnIGluICQoPC9wcm9jL2NtZGxpbmUp
OyBkbwogICAgaWYgW1sgJGFyZyA9fiBec3lzdGVtZC5jcHVfYWZmaW5pdHk9IF1dOyB0aGVuCiAgIC
AgIGVjaG8gJHthcmcjKj19CiAgICAgIHJldHVybiAwCiAgICBmaQogIGRvbmUKICByZXR1cm4gMQp9Cg
pnZXRDUFVDb3VudCAoKSB7CiAgbG9jYWwgY3B1c2V0PSIkMSIKICBsb2NhbCBjcHVsaXN0PSgpCiA
gbG9jYWwgY3B1cz0wCiAgbG9jYWwgbWluY3B1cz0yCgogIGlmIFtbIC16ICRjcHVzZXQgfHwgJGNwdX
NldCA9fiBbXjAtOSwtXSBdXTsgdGhlbgogICAgZWNobyAkbWluY3B1cwogICAgcmV0dXJuIDEKICBma
QoKICBJRlM9JywnIHJlYWQgLXJhIGNwdWxpc3QgPDw8ICRjcHVzZXQKCiAgZm9yIGVsbSBpbiAiJHtj
cHVsaXN0W0BdfSI7IGRvCiAgICBpZiBbWyAkZWxtID1+IF5bMC05XSskIF1dOyB0aGVuCiAgICAgICgo
IGNwdXMrKyApKQogICAgZWxpZiBbWyAkZWxtID1+IF5bMC05XSstWzAtOV0rJCBdXTsgdGhlbgogIC
AgICBsb2NhbCBsb3c9MCBoaWdoPTAKICAgICAgSUZTPSctJyByZWFkIGxvdyBoaWdoIDw8PCAkZW
xtCiAgICAgICgoIGNwdXMgKz0gaGlnaCAtIGxvdyArIDEgKSkKICAgIGVsc2UKICAgICAgZWNobyAkbWl
uY3B1cwogICAgICByZXR1cm4gMQogICAgZmkKICBkb25lCgogICMgUmV0dXJuIGEgbWluaW11bSBv
ZiAyIGNwdXMKICBlY2hvICQoKCBjcHVzID4gJG1pbmNwdXMgPyBjcHVzIDogJG1pbmNwdXMgKSkKI
CByZXR1cm4gMAp9CgpyZXNldE9WU3RocmVhZHMgKCkgewogIGxvY2FsIGNwdWNvdW50PSIkMSI
KICBsb2NhbCBjdXJSZXZhbGlkYXRvcnM9MAogIGxvY2FsIGN1ckhhbmRsZXJzPTAKICBsb2NhbCBkZ
XNpcmVkUmV2YWxpZGF0b3JzPTAKICBsb2NhbCBkZXNpcmVkSGFuZGxlcnM9MAogIGxvY2FsIHJjP
TAKCiAgY3VyUmV2YWxpZGF0b3JzPSQocHMgLVRlbyBwaWQsdGlkLGNvbW0sY21kIHwgZ3JlcCAtZ
SByZXZhbGlkYXRvciB8IGdyZXAgLWMgb3ZzLXZzd2l0Y2hkKQogIGN1ckhhbmRsZXJzPSQocHMgLV
RlbyBwaWQsdGlkLGNvbW0sY21kIHwgZ3JlcCAtZSBoYW5kbGVyIHwgZ3JlcCAtYyBvdnMtdnN3aXRja
GQpCgogICMgQ2FsY3VsYXRlIHRoZSBkZXNpcmVkIG51bWJlciBvZiB0aHJlYWRzIHRoZSBzYW1lIHd
heSBPVlMgZG9lcy4KICAjIE9WUyB3aWxsIHNldCB0aGVzZSB0aHJlYWQgY291bnQgYXMgYSBvbmUg
c2hvdCBwcm9jZXNzIG9uIHN0YXJ0dXAsIHNvIHdlCiAgIyBoYXZlIHRvIGFkanVzdCB1cCBvciBkb3duIG
R1cmluZyB0aGUgYm9vdCB1cCBwcm9jZXNzLiBUaGUgZGVzaXJlZCBvdXRjb21lIGlzCiAgIyB0byBub3
QgcmVzdHJpY3QgdGhlIG51bWJlciBvZiB0aHJlYWQgYXQgc3RhcnR1cCB1bnRpbCB3ZSByZWFjaCB
hIHN0ZWFkeQogICMgc3RhdGUuICBBdCB3aGljaCBwb2ludCB3ZSBuZWVkIHRvIHJlc2V0IHRoZXNlI
GJhc2VkIG9uIG91ciByZXN0cmljdGVkICBzZXQKICAjIG9mIGNvcmVzLgogICMgU2VlIE9WUyBmdW5jd
GlvbiB0aGF0IGNhbGN1bGF0ZXMgdGhlc2UgdGhyZWFkIGNvdW50czoKICAjIGh0dHBzOi8vZ2l0aHVi
LmNvbS9vcGVudnN3aXRjaC9vdnMvYmxvYi9tYXN0ZXIvb2Zwcm90by9vZnByb3RvLWRwaWYtdXBjY
WxsLmMjTDYzNQogICCgoIGRlc2lyZWRSZXZhbGlkYXRvcnM9JGNwdWNvdW50IC8gNCArIDEgKSkKI
CAoKCBkZXNpcmVkSGFuZGxlcnM9JGNwdWNvdW50IC0gJGRlc2lyZWRSZXZhbGlkYXRvcnMgKSkK
CgogIGlmIFtbICRjdXJSZXZhbGlkYXRvcnMgLW5lICRkZXNpcmVkUmV2YWxpZGF0b3JzIHx8ICRjdXJI
YW5kbGVycyAtbmUgJGRlc2lyZWRIYW5kbGVycyBdXTsgdGhlbgoKICAgIGxvZ2dlciAiUmVjb3Zlcnk6IF
lLXNldHRpbmcgT1ZTIHJldmFsaWRhdG9yIHRocmVhZHM6ICR7Y3VyUmV2YWxpZGF0b3JzfSAtPiAk
e2Rlc2lyZWRSZXZhbGlkYXRvcnN9IgogICAgbG9nZ2VyICJSZWNvdmVyeTogUmUtc2V0dGluZyBPVl
MgaGFuZGxlciB0aHJlYWRzOiAke2N1ckhhbmRsZXJzfSAtPiAke2Rlc2lyZWRIYW5kbGVyc30iCgogICA
b3ZzLXZzY3RsIHNldCBcCiAgICAgICAgIE9wZW5fdlN3aXRjaCAuIFwKICAgICAgb3RoZXItY29uZmlnOm4ta
GFuZGxlci10aHJlYWRzPSR7ZGVzaXJlZEhhbmRsZXJzfSBcCiAgICAgICAgIG90aGVyLWNvbmZpZzpuLXJl
dmFsaWRhdG9yLXRocmVhZHM9JHtkZXNpcmVkUmV2YWxpZGF0b3JzfQogICAgcmM9JD8KICBma
QoKICByZXR1cm4gJHJjCn0KCnJlc2V0QWZmaW5pdHkoKSB7CiAgbG9jYWwgY3B1c2V0PSIkMSIKIC
Bsb2NhbCBmYWlsY291bnQ9MAogIGxvY2FsIHN1Y2Nlc3Njb3VudD0wCiAgbG9nZ2VyICJSZWNvdmV
yeTogU2V0dGluZyBDUFUgYWZmaW5pdHkgZm9yIGNyaXRpY2FsIHByb2Nlc3NlcyBcIiRDUklUSUNB
TF9QUk9DRVNTRVNcIiB0byAkY3B1c2V0IgogIGZvciBwcm9jIGluICRDUklUSUNBTF9QUk9DRVNTR
VM7IGRvCiAgICBsb2NhbCBwaWRzPSIkKHBncmVwICRwcm9jKSIKICAgIGZvciBwaWQgaW4gJHBpZ
HM7IGRvCiAgICAgIGxvY2FsIHRhc2tzZXRPdXRwdXQKICAgICAgdGFza3NldE91dHB1dD0iJCh0YXN
rc2V0IC1hcGMgIiRjcHVzZXQiICRwaWQgMj4mMSkiCiAgICAgIGlmIFtbICQ/IC1uZSAwIF1dOyB0aGVu
CiAgICAgICAgZWNobyAiRVJST1I6ICR0YXNrc2V0T3V0cHV0IgogICAgICAgICgoZmFpbGNvdW50Kys
pKQogICAgICBlbHNlCiAgICAgICAgKChzdWNjZXNzY291bnQrKykpCiAgICAgIGZpCiAgICBkb25lCiAgZ
G9uZQoKICByZXNldE9WU3RocmVhZHMgIiQoZ2V0Q1BVQ291bnQgJHtjcHVzZXR9KSIKICBpZiBbW
yAkPyAtbmUgMCBdXTsgdGhlbgogICAgKChmYWlsY291bnQrKykpCiAgZWxzZQogICAgKChzdWNjZX
NzY291bnQrKykpCiAgZmkKCiAgbG9nZ2VyICJSZWNvdmVyeTogUmUtYWZmaW5lZCAkc3VjY2Vzc2N2N
vdW50IHBpZHMgc3VjY2Vzc2Z1bGx5IgogIGlmIFtbICRmYWlsY291bnQgLWd0IDAgXV07IHRoZW4KIC
AgIGxvZ2dlciAiUmVjb3Zlcnk6IEZhaWxlZCB0byByZS1hZmZpbmUgJGZhaWxjb3VudCBwcm9jZXNzZX
MiCiAgICByZXR1cm4gMQogIGZpCn0KCnNldFVucmVzdHJpY3RlZCgpIHsKICBsb2dnZXIgIlJlY292ZXJ

5OiBTZXR0aW5nIGNyaXRpY2FsIHN5c3RlbSBwcm9jZXNzZXMgdG8gaGF2ZSB1bnJlc3RyaWN0ZW
QgQ1BVIGFjY2VzcyIKICByZXNldEFmZmluaXR5ICIkKHVucmVzdHJpY3RlZENwdXNldCkiCn0KCnNld
FJlc3RyaWN0ZWQoKSB7CiAgbG9nZ2VyICJSZWNvdmVyeTogUmVzZXR0aW5nIGNyaXRpY2FsIHN
5c3RlbSBwcm9jZXNzZXMgYmFjayB0byBub3JtYWxseSByZXN0cmljdGVkIGFjY2VzcyIKICByZXNldEFm
ZmluaXR5ICIkKHJlc3RyaWN0ZWRDcHVzZXQpIgp9CgpjdXJyZW50QWZmaW5pdHkoKSB7CiAgbG9j
YWwgcGlkPSIkMSIKICB0YXNrc2V0IC1wYyAkcGlkIHwgYXdrIC1GJzogJyAne3ByaW50ICQyfScKfQoK
d2l0aGluKCkgewogIGxvY2FsIGxhc3Q9JDEgY3VycmVudD0kMiB0aHJlc2hvbGQ9JDMKICBsb2NhbCB
kZWx0YT0wIHBjaGFuZ2UKICBkZWx0YT0kKCggY3VycmVudCAtIGxhc3QgKSkKICBpZiBbWyAkY3Vy
cmVudCAtZXEgJGxhc3QgXV07IHRoZW4KICAgIHBjaGFuZ2U9MAogIGVsaWYgW1sgJGxhc3QgLWV
xIDAgXV07IHRoZW4KICAgIHBjaGFuZ2U9MTAwMDAwMAogIGVsc2UKICAgIHBjaGFuZ2U9JCgoICg
gJGRlbHRhICogMTAwKSAvIGxhc3QgKSkKICBmaQogIGVjaG8gLW4gImxhc3Q6JGxhc3QgY3VycmV
udDokY3VycmVudCBkZWx0YTokZGVsdGEgcGNoYW5nZToke3BjaGFuZ2V9JTogIgogIGxvY2FsIGFic
29sdXRlIGxpbWl0CiAgY2FzZSAkdGhyZXNob2xkIGluCiAgICAqJSkKICAgICAgYWJzb2x1dGU9JHtwY2
hhbmdlIyMtfSAjIGFic29sdXRlIHZhbHVlCiAgICAgIGxpbWl0PSR7dGhyZXNob2xkJSUlfQogICAgICA7O
wogICAgKikKICAgICAgYWJzb2x1dGU9JHtkZWx0YSMjLX0gIyBhYnNvbHV0ZSB2YWx1ZQogICAgICB
saW1pdD0kdGhyZXNob2xkCiAgICAgIDs7CiAgZXNhYwogIGlmIFtbICRhYnNvbHV0ZSAtbGUgJGxpb
Wl0IF1dOyB0aGVuCiAgICBlY2hvICJ3aXRoaW4gKCsvLSkkdGhyZXNob2xkIgogICAgcmV0dXJuIDAKI
CBlbHNlCiAgICBlY2hvICJvdXRzaWRlICgrLy0pJHRocmVzaG9sZCIKICAgIHJldHVybiAxCiAgZmkKfQok
c3RlYWR5c3RhdGUoKSB7CiAgbG9jYWwgbGFzdD0kMSBjdXJyZW50PSQyCiAgaWYgW1sgJGxhc3Q
gLWx0ICRTTEVBRFlfU1RBVEVfTUlOSU1VTSBdXTsgdGhlbgogICAgZWNobyAibGFzdDokbGFzdCBj
dXJyZW50OiRjdXJyZW50IFdhaXRpbmcgdG8gcmVhY2ggJFNURUFEWV9TVEFURV9NSU5JTVVNIG
JlZm9yZSBjaGVja2luZyBmb3Igc3RlYWR5LXN0YXRlIgogICAgcmV0dXJuIDEKICBmaQogIHdpdGhpbiA
kbGFzdCAkY3VycmVudCAkU1RFQURZX1NUQVRFX1RIUkVTSE9MRAp9Cgp3YWl0Rm9yUmVhZHk
oKSB7CiAgbG9nZ2VyICJSZWNvdmVyeTogV2FpdGluZyAke01BWElNVU1fV0FJVF9USU1FfXMgZm9
yIHRoZSBpbml0aWFsaXphdGlvbiB0byBjb21wbGV0ZSIKICBsb2NhbCBsYXN0U3lzdGVtZENwdXNldD
0iJChjdXJyZW50QWZmaW5pdHkgMSkiCiAgbG9jYWwgbGFzdERlc2lyZWRDcHVzZXQ9IiQodW5yZXN
0cmljdGVkQ3B1c2V0KSIKICBsb2NhbCB0PTAgcz0xMAogIGxvY2FsIGxhc3RDY291bnQ9MCBjY291bn
Q9MCBzdGVhZHlTdGF0ZVRpbWU9MAogIHdoaWxlIFtbICR0IC1sdCAkTUFYSU1VTV9XQUlUX1RJT
UUgXV07IGRvCiAgICBzbGVlcCAkcwogICAgKCh0ICs9IHMpKQogICAgIyBSZS1jaGVjayB0aGUgY3Vy
cmVudCBhZmZpbml0eSBvZiBzeXN0ZW1kLCBpbiBjYXNlIHNvbWUgb3RoZXIgcHJvY2VzcyBoYXMgY2
hhbmdlZCBpdAogICAgbG9jYWwgc3lzdGVtZENwdXNldD0iJChjdXJyZW50QWZmaW5pdHkgMSkiCiAg
ICAjIFJlLWNoZWNrIHRoZSB1bnJlc3RyaWN0ZWQgQ3B1c2V0LCBhcyB0aGUgYWxsb3dlZCBzZXQgb
2YgdW5yZXNlcnZlZCBjb3JlcyBtYXkgY2hhbmdlIGFzIHBvZHMgYXJlIGFzc2lnbmVkIHRvIGNvcmVzCiAg
ICBsb2NhbCBkZXNpcmVkQ3B1c2V0PSIkKHVucmVzdHJpY3RlZENwdXNldCkiCiAgICBpZiBbWyAkc3l
dGVtZENwdXNldCAhPSAkbGFzdFN5c3RlbWRDcHVzZXQgfHwgJGxhc3REZXNpcmVkQ3B1c2V0ICE
9ICRkZXNpcmVkQ3B1c2V0IF1dOyB0aGVuCiAgICAgICAgIHJlc2V0QWZmaW5pdHkgIiRkZXNpcmVkQ3B1
c2V0IgogICAgICBsYXN0U3lzdGVtZENwdXNldD0iJChjdXJyZW50QWZmaW5pdHkgMSkiCiAgICAgIGx
hc3REZXNpcmVkQ3B1c2V0PSIkZGVzaXJlZENwdXNldCIKICAgIGZpCgogICAgIyBEZXRlY3Qgc3RlY
WR5LXN0YXRlIHBvZCBjb3VudAogICAgY2NvdW50PSQoY3JpY3RsIHBzIHwgd2MgLWwpCiAgICBpZi
BzdGVhZHlzdGF0ZSAkbGFzdENjb3VudCAkY2NvdW50OyB0aGVuCiAgICAgICgoc3RlYWR5U3RhdG
VUaW1lICs9IHMpKQogICAgICBlY2hvICJTdGVhZHktc3RhdGUgZm9yICR7c3RlYWR5U3RhdGVUaW
1lfXMvJHtTVEVBRFlfU1RBVEVfV0lORE9XfXMiCiAgICAgIGlmIFtbICRzdGVhZHlTdGF0ZVRpbWUgL
WdlICRTVEVBRFlfU1RBVEVfV0lORE9XIF1dOyB0aGVuCiAgICAgICAgbG9nZ2VyICJSZWNvdmVyeT
ogU3RlYWR5LXN0YXRlICgrLy0gJFNURUFEWV9TVEFURV9USFJFU0hPTEQpIGZvciAke1NURUFE
WV9TVEFURV9XSU5ET1d9czogRG9uZSIKICAgICAgICByZXR1cm4gMAogICAgICBmaQogICAgZWx
zZQogICAgICBpZiBbWyAkc3RlYWR5U3RhdGVUaW1lIC1ndCAwIF1dOyB0aGVuCiAgICAgICAgZWN
obyAiUmVzZXR0aW5nIHN0ZWFkeS1zdGF0ZSB0aW1lciIKICAgICAgICBzdGVhZHlTdGF0ZVRpbWU
9MAogICAgICBmaQogICAgZmkKICAgIGxhc3RDY291bnQ9JGNjb3VudAogIGRvbmUKICBsb2dnZXIgI
lJlY292ZXJ5OiBSZWNvdmVyeSBDb21wbGV0ZSBUaW1lb3V0Igp9CgptYWluKCkgewogIGlmICEgdW5
yZXN0cmljdGVkQ3B1c2V0ID4mL2Rldi9udWxsOyB0aGVuCiAgICBsb2dnZXIgIlJlY292ZXJ5OiBObyB1b
nJlc3RyaWN0ZWQgQ3B1c2V0IGNvdWxkIGJlIGRldGVjdGVkIgogICAgcmV0dXJuIDEKICBmaQoKICB
pZiAhIHJlc3RyaWN0ZWRDcHVzZXQgPiYvZGV2L251bGw7IHRoZW4KICAgIGxvZ2dlciAiUmVjb3Zlcnk
6IE5vIHJlc3RyaWN0ZWQgQ3B1c2V0IGhhcyBiZWVuIGNvbmZpZ3VyZWQuICBXZSBhcmUgYWxyZW
FkeSBydW5uaW5nIHVucmVzdHJpY3RlZC4iCiAgICByZXR1cm4gMAogIGZpCgogICMgRW5zdXJlIHdlI
HJlc2V0IHRoZSBDUFUgYWZmaW5pdHkgd2hlbiB3ZSBleGl0IHRoaXMgc2NyaXB0IGZvciBhbnkgcmVh
c29uCiAgIyBUaGlzIHdheSBlaXRoZXIgYWZ0ZXIgdGhlIHRpbWVyIGV4cGlyZXMgb3IgYWZ0ZXIgdGhl

```
HByb2Nlc3MgaXMgaW50ZXJydXB0ZWQKICAjIHZpYSBeQyBvciBTSUdURVJNLCB3ZSByZXR1cm4g
dGhpbmdzIGJhY2sgdG8gdGhlIHdheSB0aGV5IHNob3VsZCBiZS4KICB0cmFwIHNldFJlc3RyaWN0ZW
QgRVhJVAoKICBsb2dnZXIgIlJlY292ZXJ5OiBSZWNvdmVyeSBNb2RlIFN0YXJ0aW5nIgoIHNldFVucm
VzdHJpY3RlZAogIHdhaXRGb3JSZWFkeQp9CgppZiBbWyAiJHtCQVNIX1NPVVJDRVswXX0iID0gIiR7
MH0iIF1dOyB0aGVuCiAgbWFpbiAiJHtAfSIKICBleGl0ICQ/CmZpCg==
        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
```

19.6.5.5. 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 で有効になります。

推奨される kdump 設定

```
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
```

19.6.6. 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、Performance プロファイル CR を使用して、単一ノードの OpenShift クラスターの UEFI セキュアブートを設定します。詳細は、パフォーマンスプロファイル を参照してください。

19.6.6.1. 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 YAML は、これらの CR をまとめたものです。

推奨される Operator Namespace および OperatorGroup 設定

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

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

19.6.6.2. 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

推奨される 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" (1)
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual (2)
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
---
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
---
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
```

(1) Operator を取得するチャネルを指定します。**stable** チャンネルが推奨されます。

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

19.6.6.3. 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 YAML の例は、必要な **ClusterLogging** および **ClusterLogForwarder** CR を示しています。

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 (1)
metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    type: "curator"
    curator:
      schedule: "30 3 * * *"
  managementState: Managed
---
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder (2)
metadata:
  name: instance
  namespace: openshift-logging
spec:
  inputs:
    - infrastructure: {}
      name: infra-logs
  outputs:
    - name: kafka-open
      type: kafka
      url: tcp://10.46.55.190:9092/test (3)
  pipelines:
    - inputRefs:
      - audit
      name: audit-logs
      outputRefs:
      - kafka-open
    - inputRefs:
      - infrastructure
      name: infrastructure-logs
      outputRefs:
      - kafka-open
```

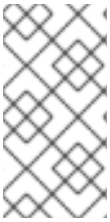
(1) 既存の **ClusterLogging** インスタンスを更新するか、存在しない場合はインスタンスを作成します。

(2) 既存の **ClusterLogForwarder** インスタンスを更新するか、存在しない場合はインスタンスを作成します。

(3) ログの転送先の Kafka サーバーの URL を指定します。

19.6.6.4. 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 では、これらの機能は Node Tuning Operator の一部です。

次の **PerformanceProfile** CR の例は、必要な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 (1)
spec:
  additionalKernelArgs:
    - rcupdate.rcu_normal_after_boot=0
    - "efi=runtime" (2)
  cpu:
    isolated: 2-51,54-103 (3)
    reserved: 0-1,52-53 (4)
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 32 (5)
        size: 1G (6)
        node: 1 (7)
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true (8)
```

(1) **name** の値が、**TunedPerformancePatch.yaml** の **spec.profile.data** フィールドと **validatorCRs/informDuValidator.yaml** の **status.configuration.source.name** フィールドで指定された値と一致することを確認します。

(2) (3) クラスターホストの UEFI セキュアブートを設定します。

(4) 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。

(5) 予約済みの CPU を設定します。ワークロードの分割が有効になっている場合、システムプロセス、カーネルスレッド、およびシステムコンテナースレッドは、これらの CPU に制限されます。分離されていないすべての CPU を予約する必要があります。

(6) Huge Page の数を設定します。

7 Huge Page のサイズを設定します。

8 リアルタイム Linux カーネルをインストールするには、**enabled** を **true** に設定します。

19.6.6.5. 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: du-ptp-slave
  namespace: openshift-ptp
spec:
  profile:
    - interface: ens5f0    1
      name: slave
      phc2sysOpts: -a -r -n 24
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison ieee1588
        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 1
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
      ptp4lOpts: -2 -s --summary_interval -4
recommend:
  - match:
      - nodeLabel: node-role.kubernetes.io/master
    priority: 4
    profile: slave
```

1 PTP クロック信号を受信するために使用されるインターフェイスを設定します。

19.6.6.6. 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:
    - data: |
        [main]
        summary=Configuration changes profile inherited from performance created tuned
        include=openshift-node-performance-openshift-node-performance-profile
        [bootloader]
        cmdline_crash=nohz_full=2-51,54-103
        [sysctl]
        kernel.timer_migration=1
        [scheduler]
        group.ice-ptp=0:f:10:*:ice-ptp.*
        [service]
        service.stalld=start,enable
        service.chronyd=stop,disable
      name: performance-patch
  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: master
      priority: 19
      profile: performance-patch
```

19.6.6.7. SR-IOV

シングルルート I/O 仮想化 (SR-IOV) は、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一般的に使用されます。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。

推奨される SR-IOV 設定

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    node-role.kubernetes.io/master: ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-mh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_mh
  vlan: 150 (1)
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-mh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci (2)
  isRdma: false
  nicSelector:
    pfNames:
      - ens7f0 (3)
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 (4)
  priority: 10
  resourceName: du_mh
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-fh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_fh
```

```
  vlan: 140 5
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-fh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 6
  isRdma: true
  nicSelector:
    pfNames:
      - ens5f0 7
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 8
  priority: 10
  resourceName: du_fh
```

1. ミッドホールネットワークの VLAN を指定します。
2. 必要に応じて、**vfio-pci** または **netdevice** のいずれかを選択します。
3. ミッド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。
4. ミッドホールネットワークの VF の数を指定します。
5. フロントホールネットワークの VLAN。
6. 必要に応じて、**vfio-pci** または **netdevice** のいずれかを選択します。
7. 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。
8. フロントホールネットワークの VF の数を指定します。

19.6.6.8. Console Operator

console-operator は、Web コンソールをクラスターにインストールして保守します。ノードが集中管理されている場合、Operator は不要であり、アプリケーションのワークロード用のスペースを確保します。次の **Console** カスタムリソース (CR) の例では、コンソールを無効にします。

推奨されるコンソール設定

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "false"
    include.release.openshift.io/self-managed-high-availability: "false"
    include.release.openshift.io/single-node-developer: "false"
    release.openshift.io/create-only: "true"
  name: cluster
spec:
```

```
  logLevel: Normal
  managementState: Removed
  operatorLogLevel: Normal
```

19.6.6.9. Grafana と 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。次の **ConfigMap** カスタムリソース (CR) は、Grafana と Alertmanager を無効にします。

推奨されるクラスター監視設定

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

19.6.6.10. ネットワーク診断

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

推奨されるネットワーク診断設定

```
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

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

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

関連情報

- 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。

19.7.1. vDU クラスターホストの推奨ファームウェア設定

OpenShift Container Platform 4.10 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。

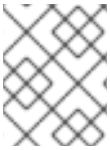
注記

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19.7 推奨されるクラスターホストファームウェア設定

ファームウェア設定	設定	説明
HyperTransport (HT)	有効	HyperTransport (HT) バスは、AMD が開発したバス技術です。HT は、ホストメモリー内のコンポーネントと他のシステムペリフェラル間の高速リンクを提供します。
UEFI	有効	vDU ホストの UEFI からの起動を有効にします。
CPU パワーとパフォーマンスポリシー	パフォーマンス	CPU パワーとパフォーマンスポリシーを設定し、エネルギー効率よりもパフォーマンスを優先してシステムを最適化します。
Uncore Frequency Scaling	Disabled	Uncore Frequency Scaling を無効にして、CPU のコア以外の部分の電圧と周波数が個別に設定されるのを防ぎます。
Uncore Frequency	最大	キャッシュやメモリーコントローラーなど、CPU のコア以外の部分を可能な最大動作周波数に設定します。
パフォーマンスの制限	Disabled	プロセッサーの Uncore Frequency 調整を防ぐために、パフォーマンス P 制限を無効にします。
強化された Intel® SpeedStep テクノロジー	有効	Enhanced Intel SpeedStep を有効にして、システムがプロセッサーの電圧とコア周波数を動的に調整できるようにし、ホストの消費電力と発熱を減らします。
Intel® Turbo Boost Technology	有効	Intel ベースの CPU で Turbo Boost Technology を有効にすると、プロセッサーコアが電力、電流、および温度の仕様制限を下回って動作している場合、自動的に定格動作周波数よりも高速に動作できるようにします。

ファームウェア設定	設定	説明
Intel Configurable TDP	有効	CPU の Thermal Design Power (TDP) を有効にします。
設定可能な TDP レベル	レベル 2	TDP レベルは、特定のパフォーマンス評価に必要な CPU 消費電力を設定します。TDP レベル 2 は、消費電力を犠牲にして、CPU を最も安定したパフォーマンスレベルに設定します。
energy Efficient Turbo	Disabled	Energy Efficient Turbo を無効にして、プロセッサーがエネルギー効率ベースのポリシーを使用しないようにします。
Hardware P-States	Disabled	**P-states** (パフォーマンスステート) を無効にして、オペレーティングシステムと CPU を最適化し、電力消費に対するパフォーマンスを向上させます。
Package C-State	C0/C1 の状態	C0 または C1 状態を使用して、プロセッサーを完全にアクティブな状態 (C0) に設定するか、ソフトウェアで実行されている CPU 内部クロックを停止します (C1)。
C1E	Disabled	CPU Enhanced Halt (C1E) は、Intel チップの省電力機能です。C1E を無効にすると、非アクティブ時にオペレーティングシステムが停止コマンドを CPU に送信することを防ぎます。
Processor C6	Disabled	C6 節電は、アイドル状態の CPU コアとキャッシュを自動的に無効にする CPU 機能です。C6 を無効にすると、システムパフォーマンスが向上します。
サブ NUMA クラスタリング	Disabled	サブ NUMA クラスタリングは、プロセッサーコア、キャッシュ、およびメモリーを複数の NUMA ドメインに分割します。このオプションを無効にすると、レイテンシーの影響を受けやすいワークロードのパフォーマンスが向上します。



注記

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

19.7.2. 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 4.10 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。

19.7.2.1. 推奨されるクラスター MachineConfig CR

次の **MachineConfig** CR は、クラスターホストを設定します。

表19.8 推奨される MachineConfig CR

CR ファイル名	説明
02-workload-partitioning.yaml	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。
MachineConfigSctp.yaml	SCTP カーネルモジュールをロードします。この **MachineConfig** CR はオプションであり、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。
MachineConfigContainerMountNS.yaml	コンテナーマウント namespace と kubelet conf を設定します。
MachineConfigAcceleratedStartup.yaml	クラスターの高速スタートアップを設定します。
06-kdump-master.yaml, **06-kdump-worker.yaml**	クラスターの **kdump** を設定します。

19.7.2.2. 推奨されるクラスター Operator

次の Operator は、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となります。

- 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

19.7.2.3. 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。また、クラスター内で以下の設定が適用されていることを確認する必要があります。

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:
  - "idle=poll"
  - "rcupdate.rcu_normal_after_boot=0"
  - "efi=runtime"
```

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
      # And the cmdline_crash CPU set must match the 'isolated' set in the associated
PerformanceProfile
      data: |
        [main]
        summary=Configuration changes profile inherited from performance created tuned
        include=openshift-node-performance-openshift-node-performance-profile
        [bootloader]
        cmdline_crash=nohz_full=2-51,54-103 (1)
        [sysctl]
        kernel.timer_migration=1
        [scheduler]
        group.ice-ptp=0:f:10:*:ice-ptp.*
        [service]
        service.stalld=start,enable
        service.chronyd=stop,disable
```

(1) リスト表示される CPU は、ホストハードウェア設定、特にシステムで使用可能な CPU の数と CPU トポロジーによって異なります。

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

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

出力例

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

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

注記

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

検証

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

1. クラスターノードへのリモートシェル接続を開きます。

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

2. リアルタイムカーネルバージョンを確認します。

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

出力例

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

19.7.3. 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 4.10 クラスターにデプロイするために必要なさまざまな設定を確認する方法について説明します。

前提条件

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

手順

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

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

出力例

```
spec:
  disableAllDefaultSources: true
```

2. 次のコマンドを実行して、必要なすべての **CatalogSource** リソースにワークロードのパーティショニング (**PreferredDuringScheduling**) のアノテーションが付けられていることを確認します。

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

出力例

```
certified-operators -- {"effect": "PreferredDuringScheduling"}
community-operators -- {"effect": "PreferredDuringScheduling"}
ran-operators 1
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

1 アノテーションが付けられていない **CatalogSource** リソースも返されます。この例では、**ran-operators CatalogSource** リソースにはアノテーションが付けられておらず、**PreferredDuringScheduling** アノテーションがありません。

注記

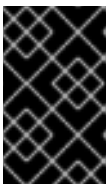
適切に設定された vDU クラスターでは、単一のアノテーション付きカタログソースのみがリスト表示されます。

3. 該当するすべての OpenShift Container Platform Operator の namespace がワークロードのパーティショニング用にアノテーションされていることを確認します。これには、コア OpenShift Container Platform とともにインストールされたすべての Operator と、参照 DU チューニング設定に含まれる追加の Operator のセットが含まれます。以下のコマンドを実行します。

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

出力例

```
default --
openshift-apiserver -- management
openshift-apiserver-operator -- management
openshift-authentication -- management
openshift-authentication-operator -- management
```



重要

追加の Operator は、ワークロードパーティショニングのためにアノテーションを付けてはなりません。前のコマンドからの出力では、追加の Operator が **--** セパレーターの右側に値なしでリストされている必要があります。

4. **ClusterLogging** 設定が正しいことを確認してください。以下のコマンドを実行します。

a. 適切な入力ログと出力ログが設定されていることを確認します。

```
$ oc get -n openshift-logging ClusterLogForwarder instance -o yaml
```

出力例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  creationTimestamp: "2022-07-19T21:51:41Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "1030342"
  uid: 8c1a842d-80c5-447a-9150-40350bdf40f0
spec:
  inputs:
  - infrastructure: {}
    name: infra-logs
  outputs:
  - name: kafka-open
    type: kafka
    url: tcp://10.46.55.190:9092/test
  pipelines:
  - inputRefs:
    - audit
    name: audit-logs
    outputRefs:
    - kafka-open
  - inputRefs:
    - infrastructure
    name: infrastructure-logs
    outputRefs:
    - kafka-open
...
```

b. キュレーションスケジュールがアプリケーションに適していることを確認します。

```
$ oc get -n openshift-logging clusterloggings.logging.openshift.io instance -o yaml
```

出力例

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  creationTimestamp: "2022-07-07T18:22:56Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "235796"
  uid: ef67b9b8-0e65-4a10-88ff-ec06922ea796
spec:
  collection:
    logs:
```

```
      fluentd: {}
      type: fluentd
  curation:
    curator:
      schedule: 30 3 * * *
    type: curator
  managementState: Managed
...
```

5. 次のコマンドを実行して、Web コンソールが無効になっている (**managementState: Removed**) ことを確認します。

```
$ oc get consoles.operator.openshift.io cluster -o jsonpath="{ .spec.managementState }"
```

出力例

```
Removed
```

6. 次のコマンドを実行して、クラスターノードで **chronyd** が無効になっていることを確認します。

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

ノードで **chronyd** のステータスを確認します。

```
sh-4.4# chroot /host
```

```
sh-4.4# systemctl status chronyd
```

出力例

```
● chronyd.service - NTP client/server
   Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset:
enabled)
   Active: inactive (dead)
     Docs: man:chronyd(8)
           man:chrony.conf(5)
```

7. **linuxptp-daemon** コンテナーへのリモートシェル接続と PTP Management Client (**pmc**) ツールを使用して、PTP インターフェイスがプライマリークロックに正常に同期されていることを確認します。

   a. 次のコマンドを実行して、**$PTP_POD_NAME** 変数に **linuxptp-daemon** Pod の名前を設定します。

```
$ PTP_POD_NAME=$(oc get pods -n openshift-ptp -l app=linuxptp-daemon -o name)
```

   b. 次のコマンドを実行して、PTP デバイスの同期ステータスを確認します。

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

出力例

```
sending: GET PORT_DATA_SET
  3cecef.fffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-1
    portState               SLAVE
    logMinDelayReqInterval  -4
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
  3cecef.fffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            3cecef.fffe.7a7020-2
    portState               LISTENING
    logMinDelayReqInterval  0
    peerMeanPathDelay       0
    logAnnounceInterval     1
    announceReceiptTimeout  3
    logSyncInterval         0
    delayMechanism          1
    logMinPdelayReqInterval 0
    versionNumber           2
```

c. 次の **pmc** コマンドを実行して、PTP クロックのステータスを確認します。

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET TIME_STATUS_NP'
```

出力例

```
sending: GET TIME_STATUS_NP
  3cecef.fffe.7a7020-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset              10 (1)
    ingress_time               1657275432697400530
    cumulativeScaledRateOffset +0.000000000
    scaledLastGmPhaseChange    0
    gmTimeBaseIndicator        0
    lastGmPhaseChange          0x0000'0000000000000000.0000
    gmPresent                  true (2)
    gmIdentity                 3c2c30.ffff.670e00
```

(1) **master_offset** は -100 から 100 ns の間である必要があります。

(2) PTP クロックがマスターに同期されており、ローカルクロックがグランドマスタークロックではないことを示します。

d. **/var/run/ptp4l.0.config** の値に対応する予期される **master offset** 値が **linuxptp-daemon-container** ログにあることを確認します。

```
$ oc logs $PTP_POD_NAME -n openshift-ptp -c linuxptp-daemon-container
```

出力例

```
phc2sys[56020.341]: [ptp4l.1.config] CLOCK_REALTIME phc offset  -1731092 s2 freq -
1546242 delay    497
ptp4l[56020.390]: [ptp4l.1.config] master offset         -2 s2 freq   -5863 path delay       541
ptp4l[56020.390]: [ptp4l.0.config] master offset         -8 s2 freq  -10699 path delay       533
```

8. 次のコマンドを実行して、SR-IOV 設定が正しいことを確認します。

   a. **SriovOperatorConfig** リソースの **disableDrain** 値が **true** に設定されていることを確認します。

   ```
   $ oc get sriovoperatorconfig -n openshift-sriov-network-operator default -o jsonpath="
   {.spec.disableDrain}{'\n'}"
   ```

   出力例

   ```
   true
   ```

   b. 次のコマンドを実行して、**SriovNetworkNodeState** 同期ステータスが **Succeeded** であることを確認します。

   ```
   $ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o jsonpath="
   {.items[*].status.syncStatus}{'\n'}"
   ```

   出力例

   ```
   Succeeded
   ```

   c. SR-IOV 用に設定された各インターフェイスの下の仮想機能 (**Vfs**) の予想される数と設定が、**.status.interfaces** フィールドに存在し、正しいことを確認します。以下に例を示します。

   ```
   $ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o yaml
   ```

   出力例

   ```
   apiVersion: v1
   items:
   - apiVersion: sriovnetwork.openshift.io/v1
     kind: SriovNetworkNodeState
   ...
     status:
       interfaces:
       ...
       - Vfs:
         - deviceID: 154c
           driver: vfio-pci
           pciAddress: 0000:3b:0a.0
           vendor: "8086"
           vfID: 0
         - deviceID: 154c
           driver: vfio-pci
   ```

```
      pciAddress: 0000:3b:0a.1
      vendor: "8086"
      vfID: 1
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.2
      vendor: "8086"
      vfID: 2
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.3
      vendor: "8086"
      vfID: 3
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.4
      vendor: "8086"
      vfID: 4
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.5
      vendor: "8086"
      vfID: 5
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.6
      vendor: "8086"
      vfID: 6
    - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:3b:0a.7
      vendor: "8086"
      vfID: 7
```

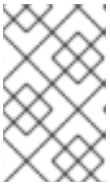
9. クラスターパフォーマンスプロファイルが正しいことを確認します。**cpu** セクションと **hugepages** セクションは、ハードウェア設定によって異なります。以下のコマンドを実行します。

```
$ oc get PerformanceProfile openshift-node-performance-profile -o yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  creationTimestamp: "2022-07-19T21:51:31Z"
  finalizers:
  - foreground-deletion
  generation: 1
  name: openshift-node-performance-profile
  resourceVersion: "33558"
  uid: 217958c0-9122-4c62-9d4d-fdc27c31118c
spec:
  additionalKernelArgs:
  - idle=poll
```

```
    - rcupdate.rcu_normal_after_boot=0
    - efi=runtime
    cpu:
      isolated: 2-51,54-103
      reserved: 0-1,52-53
    hugepages:
      defaultHugepagesSize: 1G
      pages:
      - count: 32
        size: 1G
    machineConfigPoolSelector:
      pools.operator.machineconfiguration.openshift.io/master: ""
    net:
      userLevelNetworking: true
    nodeSelector:
      node-role.kubernetes.io/master: ""
    numa:
      topologyPolicy: restricted
    realTimeKernel:
      enabled: true
  status:
    conditions:
    - lastHeartbeatTime: "2022-07-19T21:51:31Z"
      lastTransitionTime: "2022-07-19T21:51:31Z"
      status: "True"
      type: Available
    - lastHeartbeatTime: "2022-07-19T21:51:31Z"
      lastTransitionTime: "2022-07-19T21:51:31Z"
      status: "True"
      type: Upgradeable
    - lastHeartbeatTime: "2022-07-19T21:51:31Z"
      lastTransitionTime: "2022-07-19T21:51:31Z"
      status: "False"
      type: Progressing
    - lastHeartbeatTime: "2022-07-19T21:51:31Z"
      lastTransitionTime: "2022-07-19T21:51:31Z"
      status: "False"
      type: Degraded
    runtimeClass: performance-openshift-node-performance-profile
    tuned: openshift-cluster-node-tuning-operator/openshift-node-performance-openshift-node-
  performance-profile
```



注記

CPU 設定は、サーバーで使用可能なコアの数に依存し、ワークロードパーティショニングの設定に合わせる必要があります。**hugepages** の設定は、サーバーとアプリケーションに依存します。

10. 次のコマンドを実行して、**PerformanceProfile** がクラスターに正常に適用されたことを確認します。

```
$ oc get performanceprofile openshift-node-performance-profile -o jsonpath="{range
.status.conditions[*]}{ @.type }{' -- '}{@.status}{'\n'}{end}"
```

出力例

```
Available -- True
Upgradeable -- True
Progressing -- False
Degraded -- False
```

11. 次のコマンドを実行して、**Tuned** パフォーマンスパッチの設定を確認します。

```
$ oc get tuneds.tuned.openshift.io -n openshift-cluster-node-tuning-operator performance-patch -o yaml
```

出力例

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  creationTimestamp: "2022-07-18T10:33:52Z"
  generation: 1
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  resourceVersion: "34024"
  uid: f9799811-f744-4179-bf00-32d4436c08fd
spec:
  profile:
  - data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
      [bootloader]
      cmdline_crash=nohz_full=2-23,26-47 (1)
      [sysctl]
      kernel.timer_migration=1
      [scheduler]
      group.ice-ptp=0:f:10:*:ice-ptp.*
      [service]
      service.stalld=start,enable
      service.chronyd=stop,disable
    name: performance-patch
  recommend:
  - machineConfigLabels:
      machineconfiguration.openshift.io/role: master
    priority: 19
    profile: performance-patch
```

(1) **cmdline=nohz_full=** の cpu リストは、ハードウェア設定によって異なります。

12. 次のコマンドを実行して、クラスターネットワーク診断が無効になっていることを確認します。

```
$ oc get networks.operator.openshift.io cluster -o jsonpath='{.spec.disableNetworkDiagnostics}'
```

出力例

```
true
```

13. **Kubelet** のハウスキーピング間隔が、遅い速度に調整されていることを確認します。これは、**containerMountNS** マシン設定で設定されます。以下のコマンドを実行します。

```
$ oc describe machineconfig container-mount-namespace-and-kubelet-conf-master | grep OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
```

出力例

```
Environment="OPENSHIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
```

14. 次のコマンドを実行して、Grafana と **alertManagerMain** が無効になっていること、および Prometheus の保持期間が 24 時間に設定されていることを確認します。

```
$ oc get configmap cluster-monitoring-config -n openshift-monitoring -o jsonpath="{ .data.config\.yaml }"
```

出力例

```
grafana:
  enabled: false
alertmanagerMain:
  enabled: false
prometheusK8s:
   retention: 24h
```

a. 次のコマンドを使用して、Grafana および **alertManagerMain** ルートがクラスター内に見つからないことを確認します。

```
$ oc get route -n openshift-monitoring alertmanager-main
```

```
$ oc get route -n openshift-monitoring grafana
```

どちらのクエリーも **Error from server (NotFound)** メッセージを返す必要があります。

15. 次のコマンドを実行して、**PerformanceProfile**、**Tuned** performance-patch、ワークロードパーティショニング、およびカーネルコマンドライン引数のそれぞれに **reserved** として割り当てられた CPU が少なくとも 4 つあることを確認します。

```
$ oc get performanceprofile -o jsonpath="{ .items[0].spec.cpu.reserved }"
```

出力例

```
0-1,52-53
```

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。

19.8. 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。

19.8.1. ZTP GitOps パイプラインでの追加のインストールマニフェストのカスタマイズ

ゼロタッチプロビジョニング (ZTP) GitOps パイプラインのインストールフェーズに含めるための追加マニフェストのセットを定義することができます。これらのマニフェストは **SiteConfig** カスタムリソース (CR) にリンクされ、インストール時にクラスターに適用されます。インストール時に **MachineConfig** CR を含めると、インストール作業が効率的になります。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. ZTP パイプラインがクラスターインストールをカスタマイズするために使用する、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** フォルダーを持つ **/siteconfig** のサンプルを示しています。

```
siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
└── extra-manifest
    └── 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。

ZTP パイプラインは、**/extra-manifest** ディレクトリーの CR をクラスタープロビジョニング時の追加のマニフェストのデフォルトセットに追加します。

19.8.2. 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、ゼロタッチプロビジョニング (ZTP) GitOps 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 を含めたり除外したりできます。

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、ZTP パイプラインはインストール中に **/source-crs/extra-manifest** 内のすべてのファイルを適用します。**inclusionDefault** を **exclude** に設定すると、その逆になります。

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

また、いくつかのオプションのフィルタリングシナリオも説明されています。

前提条件

- 必要なインストール CR とポリシー CR を生成するためにハブクラスターを設定している。
- カスタムサイトの設定データを管理する Git リポジトリーを作成しています。リポジトリーはハブクラスターからアクセス可能で、Argo CD アプリケーションのソースリポジトリーとして定義されている必要があります。

手順

1. 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-4.10"
  sshPublicKey: "<ssh_public_key>"
  clusters:
- clusterName: "site1-sno-du"
  extraManifests:
    filter:
      exclude:
        - 03-sctp-machine-config-worker.yaml
```

   ZTP パイプラインは、インストール中に **03-sctp-machine-config-worker.yaml** CR をスキップします。**/source-crs/extra-manifest** 内の他のすべての CR が適用されます。

2. **SiteConfig** CR を保存し、変更をサイト設定リポジトリーにプッシュします。
   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 ZTP パイプラインがすべての **/source-crs/extra-manifest** CR を適用しないようにするには、**SiteConfig** CR で次の YAML を適用します。

```
- clusterName: "site1-sno-du"
  extraManifests:
    filter:
      inclusionDefault: exclude
```

4. オプション: インストール中にすべての **/source-crs/extra-manifest** RAN CR を除外し、代わりにカスタム CR ファイルを含めるには、カスタム **SiteConfig** CR を編集してカスタムマニフェストフォルダーと **include** ファイルを設定します。次に例を示します。

```
clusters:
- clusterName: "site1-sno-du"
  extraManifestPath: "<custom_manifest_folder>" (1)
  extraManifests:
    filter:
      inclusionDefault: exclude (2)
      include:
        - custom-sctp-machine-config-worker.yaml
```

(1) **<custom_manifest_folder>** を、カスタムインストール CR を含むフォルダーの名前 (**user-custom-manifest/** など) に置き換えます。

(2) インストール中に ZTP パイプラインが **/source-crs/extra-manifest** 内のファイルを適用しないようにするには、**inclusionDefault** を **exclude** に設定します。

次の例は、カスタムフォルダー構造を示しています。

```
siteconfig
  ├── site1-sno-du.yaml
  └── user-custom-manifest
      └── custom-sctp-machine-config-worker.yaml
```

19.9. 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。

19.9.1. 追加の変更のクラスターへのデプロイ

GitOps ZTP パイプラインの基本設定以外のクラスター設定の変更が必要な場合、3 つのオプションがあります。

ZTP パイプラインが完了した後に、追加設定を適用します。

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。

追加する、追加 ZTP ライブラリーにコンテンツを追加します。

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

- 追加マニフェストの追加は、ZTP GitOps 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。

19.9.2. 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** コンテナーの GitOps プラグインで提供されるベースソース CR の上に追加の設定の詳細をオーバーレイできます。**PolicyGenTemplate** CR は、ベース CR の論理マージまたはパッチとして解釈できます。**PolicyGenTemplate** CR を使用して、ベース CR の単一フィールドを更新するか、ベース CR の内容全体をオーバーレイします。ベース CR にない値の更新やフィールドの挿入が可能です。

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

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。

手順

1. 既存のコンテンツのベースラインソース CR を確認します。参考となる **PolicyGenTemplate** CR に記載されているソース CR を ZTP (zero touch provisioning) コンテナーから抽出し、確認することができます。

   a. **/out** フォルダーを作成します。

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

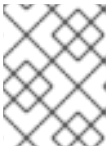
   b. ソース CR を抽出します。

   ```
   $ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v{product-version}.1 extract /home/ztp --tar | tar x -C ./out
   ```

2. **./out/source-crs/PerformanceProfile.yaml** にあるベースライン **PerformanceProfile** CR を確認します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
```

```
    - "idle=poll"
    - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
    pages:
      - size: $size
        count: $count
        node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/$mcp: ''
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```



注記

ソース CR のフィールドで **$...** を含むものは、**PolicyGenTemplate** CR で提供されない場合、生成された CR から削除されます。

3. **group-du-sno-ranGen.yaml** リファレンスファイルの **PerformanceProfile** の **PolicyGenTemplate** エントリーを更新します。次の例の **PolicyGenTemplate** CR スタンザは、適切な CPU 仕様を提供し、**hugepages** 設定を設定し、**globallyDisableIrqLoadBalancing** を false に設定する新しいフィールドを追加しています。

```
- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    name: openshift-node-performance-profile
  spec:
    cpu:
      # These must be tailored for the specific hardware platform
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
      pages:
        - size: 1G
          count: 10
    globallyDisableIrqLoadBalancing: false
```

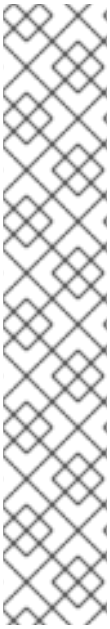
4. Git で **PolicyGenTemplate** 変更をコミットし、GitOps ZTP argo CD アプリケーションによって監視される Git リポジトリーにプッシュします。

出力例

ZTP アプリケーションは、生成された **PerformanceProfile** CR を含む RHACM ポリシーを生成しま

す。この CR の内容は， **PolicyGenTemplate** の **PerformanceProfile** エントリーから **metadata** と **spec** の内容をソース CR にマージすることで得られるものである．作成される CR には以下のコンテンツが含まれます。

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



注記

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

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

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

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

19.9.3. GitOps ZTP パイプラインへの新規コンテンツの追加

GitOps ZTP サイトジェネレーターコンテナーのソース CR は、RAN 分散ユニット (DU) アプリケー

ションの重要な機能とノードチューニング設定の一式を提供します。これらは、ZTP でデプロイするクラスターに適用されます。**ztp-site-generate** コンテナー内の既存のソース CR を追加または変更するには、**ztp-site-generate** コンテナーを再構築し、通常はハブクラスターに関連付けられた切断されたレジストリーから、ハブクラスターで利用できるようにします。有効な OpenShift Container Platform CR を追加できます。

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. 更新した **ztp-site-generate** コンテナーに含めるソース CR YAML ファイルが含まれるディレクトリーを作成します。以下に例を示します。

```
ztp-update/
├── example-cr1.yaml
├── example-cr2.yaml
└── ztp-update.in
```

2. 以下の内容を **ztp-update.in** Containerfile に追加します。

```
FROM registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.10

ADD example-cr2.yaml /kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
ADD example-cr1.yaml /kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
```

3. **ztp-update/** フォルダーでターミナルを開き、コンテナーを再ビルドします。

```
$ podman build -t ztp-site-generate-rhel8-custom:v4.10-custom-1
```

4. ビルドしたコンテナーイメージを非接続レジストリーにプッシュします。以下に例を示します。

```
$ podman push localhost/ztp-site-generate-rhel8-custom:v4.10-custom-1
registry.example.com:5000/ztp-site-generate-rhel8-custom:v4.10-custom-1
```

5. ハブクラスターの Argo CD インスタンスにパッチを適用し、新たにビルドされたコンテナーイメージを参照します。

```
$ oc patch -n openshift-gitops argocd openshift-gitops --type=json -p '[{"op": "replace",
"path":"/spec/repo/initContainers/0/image", "value": "registry.example.com:5000/ztp-site-
generate-rhel8-custom:v4.10-custom-1"} ]'
```

Argo CD インスタンスにパッチを適用すると、**openshift-gitops-repo-server** Pod は自動的に再起動します。

検証

1. 新規の **openshift-gitops-repo-server** Pod の初期化が完了し、以前のリポジトリー Pod が終了していることを確認します。

```
$ oc get pods -n openshift-gitops | grep openshift-gitops-repo-server
```

出力例

```
openshift-gitops-server-7df86f9774-db682        1/1     Running        1          28s
```

新規の **openshift-gitops-repo-server** Pod の初期化が完了し、新たに追加されたコンテナーイメージコンテンツが利用可能になる前に以前の Pod が終了するまで待機する必要があります。

関連情報

- または、パッチファイルを適用する前に、更新された **initContainer** イメージで **argocd-openshift-gitops-patch.json** を変更することにより、ArgoCD を使用したハブクラスターの設定 で説明されているように、ArgoCD インスタンスにパッチを適用できます。

19.9.4. 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

デプロイされたクラスターのゼロタッチプロビジョニング (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** ファイルに存在している必要があります。

4 **bindingExcludedRules** で定義されたラベルは 'ztp-done:' でなければなりません。**ztp-done** ラベルは、Topology Aware Lifecycle Manager と調整するために使用されます。

5 **mcp** はソースファイル **validatorCRs/informDuValidator.yaml** で使用される **MachineConfigPool** オブジェクトを定義する。これは、単一ノードの場合は **master** であり、標準のクラスターデプロイメントの場合は 3 ノードクラスターデプロイメントおよび **worker** である必要があります。

6 オプション: デフォルト値は **inform** です。

7 この値は、生成された RHACM ポリシーの名前の一部として使用されます。単一ノードの例の生成されたバリデーターポリシーは **group-du-sno-validator-du-policy** という名前です。

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

関連情報

- GitOps ZTP のアップグレード

19.9.5. PolicyGenTemplate CR を使用した PTP 高速イベントの設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイされた vRAN クラスターに PTP ファストイベントを設定することができます。**PolicyGenTemplate** のカスタムリソース (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。

前提条件

- カスタムサイトの設定データを管理する Git リポジトリーを作成している。

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"
```

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)
      ptp4lOpts: "-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 **ptp4lOpts** に **--summary_interval -4** 値を追加する必要があります。

(4) **phc2sysOpts** の値が必要です。**-m** はメッセージを **stdout** に出力します。**linuxptp-daemon DaemonSet** はログを解析し、Prometheus メトリックを生成します。

(5) オプション: **ptpClockThreshold** スタンザが存在しない場合は、**ptpClockThreshold** フィールドにデフォルト値が使用されます。スタンザは、デフォルトの **ptpClockThreshold** 値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。

3. 以下の **PolicyGenTemplate** の変更を、特定のサイトの YAML ファイル (例: **example-sno-site.yaml**) に適用してください。

   a. **.sourceFiles** に、AMQ ルーターを設定する **Interconnect** CR ファイルを **config-policy** に追加します。

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

4. 必要なその他の変更およびファイルをカスタムサイトリポジトリーにマージします。

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

関連情報

- AMQ Interconnect Operator のインストール方法に関する詳細は、AMQ メッセージングバスのインストール を参照してください。

19.9.6. PolicyGenTemplate CR を使用したベアメタルイベント監視の設定

GitOps Zero Touch Provisioning (ZTP) パイプラインを使用してデプロイされた vRAN クラスターに、ベアメタルハードウェアイベントを設定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ログインしている。
- カスタムサイトの設定データを管理する Git リポジトリーを作成している。

手順

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

```
# AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
# Bare Metal Event 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ーにプッシュして、GitOps ZTP を使用してベアメタルイベント監視を新しいサイトにデプロイします。

5. 次のコマンドを実行して Redfish シークレットを作成します。

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password> \
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

関連情報

- Bare Metal Event Relay のインストール方法に関する詳細は、CLI を使用した Bare Metal Event リレーのインストール を参照してください。

関連情報

- シークレットのユーザー名、パスワード、およびホスト IP アドレスを作成する方法の詳細は、ベアメタルイベントおよびシークレット CR の作成 を参照してください。

19.10. 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、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。

重要

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

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

関連情報

- Topology Aware Lifecycle Manager の詳細は、Topology Aware Lifecycle Manager の概要 を参照してください。

19.10.1. 切断された環境でのクラスターの更新

GitOps ZTP および Topology Aware Lifecycle Manager (TALM) を使用してデプロイした管理対象クラスターおよびマネージドクラスターの Operator をアップグレードできます。

19.10.1.1. 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、サーバーのアーキテクチャーを指定します。

      ```
      $ ARCHITECTURE=<server_architecture>
      ```

   c. 次のコマンドを実行して、Quay からリリースイメージダイジェストを取得します。

      ```
      $ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-
      release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From:
      .*@//p')"
      ```

   d. 次のコマンドを実行して、ダイジェストアルゴリズムを設定します。

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

e. 次のコマンドを実行して、ダイジェスト署名を設定します。

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```

f. 次のコマンドを実行して、mirror.openshift.com Web サイトからイメージ署名を取得します。

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature-1" | base64 -w0 && echo)
```

g. 以下のコマンドを実行して、イメージ署名を **checksum-<OCP_RELEASE_NUMBER>.yaml** ファイルに保存します。

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. 更新グラフを準備します。更新グラフを準備するオプションは 2 つあります。

a. OpenShift Update Service を使用します。
ハブクラスターでグラフを設定する方法の詳細については、 OpenShift Update Service の Operator のデプロイ および グラフデータ init コンテナーのビルド を参照してください。

b. アップストリームグラフのローカルコピーを作成します。マネージドクラスターにアクセスできる非接続環境の **http** または **https** サーバーで更新グラフをホストします。更新グラフをダウンロードするには、以下のコマンドを使用します。

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.10 -o ~/upgrade-graph_stable-4.10
```

- 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

- ZTP の更新方法の詳細は、GitOps ZTP のアップグレード を参照してください。
- OpenShift Container Platform イメージリポジトリーをミラーリングする方法の詳細は、OpenShift Container Platform イメージリポジトリーのミラーリング を参照してください。
- 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。
- 切断された環境を準備し、目的のイメージリポジトリーをミラーリングする方法に関する詳細は、非接続環境の準備 を参照してください。

- 更新チャネルとリリースの詳細は、更新チャネルとリリースについて を参照してください。

19.10.1.2. 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。
- ZTP を最新バージョンに更新します。
- ZTP を使用して 1 つ以上のマネージドクラスターをプロビジョニングします。
- 目的のイメージ リポジトリーをミラーリングします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ハブクラスターで RHACM ポリシーを作成します。

手順

1. プラットフォーム更新用の **PolicyGenTemplate** CR を作成します。
   a. 次の **PolicyGenTemplate** CR の内容を **du-upgrade.yaml** ファイルに保存します。

プラットフォーム更新の PolicyGenTemplate の例

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: ImageSignature.yaml 1
      policyName: "platform-upgrade-prep"
      binaryData:
        ${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64} 2
    - fileName: DisconnectedICSP.yaml
      policyName: "platform-upgrade-prep"
      metadata:
        name: disconnected-internal-icsp-for-ocp
      spec:
        repositoryDigestMirrors: 3
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-release
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
            source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

```
- fileName: ClusterVersion.yaml 4
  policyName: "platform-upgrade-prep"
  metadata:
    name: version
    annotations:
      ran.openshift.io/ztp-deploy-wave: "1"
  spec:
    channel: "stable-4.10"
    upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.10
- fileName: ClusterVersion.yaml 5
  policyName: "platform-upgrade"
  metadata:
    name: version
  spec:
    channel: "stable-4.10"
    upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.10
    desiredUpdate:
      version: 4.10.4
  status:
    history:
      - version: 4.10.4
        state: "Completed"
```

1. **ConfigMap** CR には、更新先の目的のリリースイメージの署名が含まれています。
2. 目的の OpenShift Container Platform リリースのイメージ署名を表示します。環境のセットアップセクションの手順に従って保存した **checksum-${OCP_RELASE_NUMBER}.yaml** ファイルから署名を取得します。
3. 目的の OpenShift Container Platform イメージを含むミラーリポジトリーを表示します。環境のセットアップセクションの手順に従って保存した **imageContentSources.yaml** ファイルからミラーを取得します。
4. アップストリームを更新する **ClusterVersion** CR を表示します。
5. 更新をトリガーする **ClusterVersion** CR を示します。イメージの事前キャッシュには、**channel**、**upstream**、および **desiredVersion** フィールドがすべて必要です。

PolicyGenTemplate CR は 2 つのポリシーを生成します。

- **du-upgrade-platform-upgrade-prep** ポリシーは、プラットフォームの更新の準備作業を行います。目的のリリースイメージシグネチャーの **ConfigMap** CR を作成し、ミラー化されたリリースイメージリポジトリーのイメージ コンテンツソースを作成し、目的の更新チャネルと切断された環境でマネージドクラスターが到達可能な更新グラフを使用してクラスターバージョンを更新します。
- **du-upgrade-platform-upgrade** ポリシーは、プラットフォームのアップグレードを実行するために使用されます。

b. **PolicyGenTemplate** CR の ZTP Git リポジトリーにある **kustomization.yaml** ファイルに **du-upgrade.yaml** ファイルの内容を追加し、変更を Git リポジトリーにプッシュします。ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

c. 以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep platform-upgrade
```

2. TALM でプラットフォームの更新を開始する前に、必要な更新リソースを適用します。

   a. 次の例に示すように、**du-upgrade-platform-upgrade-prep** ポリシーとターゲットマネージドクラスターを使用して **platform-upgrade-prep ClusterUpgradeGroup** CR の内容を **cgu-platform-upgrade-prep.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: true
```

   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-platform-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 --all-namespaces
```

3. **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、**du-upgrade-platform-upgrade** 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更新 **ClusterGroupUpdate** CR の内容を **cgu-platform-upgrade.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

   b. 次のコマンドを実行して、**ClusterGroupUpdate** CR をハブクラスターに適用します。

```
$ oc apply -f cgu-platform-upgrade.yml
```

4. 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}'
```

5. 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

- 切断された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。

19.10.1.3. 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。
- ZTP を最新バージョンに更新します。
- 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:v4.10 (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ネルが存在する場合は、それらを削除します。ZTP イメージのデフォルトのサブスクリプションチャネルが更新に使用されます。

注記

ZTP 4.10 で適用される Operator のデフォルトチャネルは **stable** ですが、**performance-addon-operator** を除きます。PAO のデフォルトのチャネルは **4.10** です。共通の **PolicyGenTemplate** CR でデフォルトのチャネルを指定することもできます。

d. **PolicyGenTemplate** CR 更新を ZTP Git リポジトリーにプッシュします。ArgoCD は Git リポジトリーから変更を取得し、ハブクラスターでポリシーを生成します。

e. 以下のコマンドを実行して、作成したポリシーを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. Operator の更新を開始する前に、必要なカタログソースの更新を適用します。

a. **operator-upgrade-prep** という名前の **ClusterGroupUpgrade** CR の内容をカタログソースポリシーと共に、ターゲットマネージドクラスターの内容を **cgu-operator-upgrade-prep.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1
```

b. 次のコマンドを実行して、ポリシーをハブ クラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

c. 更新プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. **spec.enable** フィールドを **false** に設定して、Operator 更新の **ClusterGroupUpgrade** CR を作成します。

a. 以下の例のように、Operator 更新 **ClusterGroupUpgrade** CR の内容を **du-upgrade-operator-catsrc-policy** ポリシーで保存して、共通の **PolicyGenTemplate** およびターゲットクラスターで作成されたサブスクリプションポリシーを **cgu-operator-upgrade.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
```

```
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy (1)
  - common-subscriptions-policy (2)
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1. このポリシーは、カタログソースから Operator イメージを取得するために、イメージの事前キャッシュ機能で必要になります。

2. ポリシーには Operator サブスクリプションが含まれます。ZTP を 4.9 から 4.10 にアップグレードするに従って ZTP を 4.9 から 4.10 にアップグレードした場合、すべての Operator サブスクリプションは **common-subscriptions-policy** ポリシーにグループ化されます。

注記

1 つの **ClusterGroupUpgrade** CR は、**ClusterGroupUpgrade** CR に含まれる 1 つのカタログソースからサブスクリプションポリシーで定義される必要な Operator のイメージのみを事前キャッシュできます。SRIOV-FEC Operator の例のように、目的の Operator が異なるカタログソースからのものである場合、別の **ClusterGroupUpgrade** CR を **du-upgrade-fec-catsrc-policy** および **du-upgrade-subscriptions-fec-policy** ポリシーで作成する必要があります。SRIOV-FEC Operator イメージの事前キャッシュと更新。

b. 次のコマンドを実行して、**ClusterGroupUpgrade** CR をハブクラスターに適用します。

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. オプション: Operator の更新用にイメージを事前キャッシュします。

a. イメージの事前キャッシュを開始する前に、以下のコマンドを実行して、サブスクリプションポリシーがこの時点で **NonCompliant** であることを確認します。

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

出力例

```
NAME                         REMEDIATION ACTION   COMPLIANCE STATE     AGE
common-subscriptions-policy   inform              NonCompliant         27d
```

b. 以下のコマンドを実行して、**ClusterGroupUpgrade** CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

c. プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

d. 以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうかを確認します。

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
```

出力例

```
[
  {
    "lastTransitionTime": "2022-03-08T20:49:08.000Z",
    "message": "The ClusterGroupUpgrade CR is not enabled",
    "reason": "UpgradeNotStarted",
    "status": "False",
    "type": "Ready"
  },
  {
    "lastTransitionTime": "2022-03-08T20:55:30.000Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
    "type": "PrecachingDone"
  }
]
```

5. Operator の更新を開始します。

a. 以下のコマンドを実行して **cgu-operator-upgrade ClusterGroupUpgrade** CR を有効にし、事前キャッシュを無効にして Operator の更新を開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

関連情報

- GitOps ZTP の更新に関する詳細は、GitOps ZTP のアップグレード を参照してください。

19.10.1.4. プラットフォームと Operator の更新を一緒に実行する

プラットフォームと Operator の更新を同時に実行できます。

前提条件

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

手順

1. プラットフォーム更新の実行および Operator 更新の実行セクションで説明されている手順に従って、更新用の **PolicyGenTemplate** CR を作成します。
2. プラットフォームの準備作業と Operator の更新を適用します。
   a. プラットフォームの更新の準備作業、カタログ ソースの更新、およびターゲット クラスターのポリシーを **含む ClusterGroupUpgrade** CR の内容を **cgu-platform-operator-upgrade-prep.yml** ファイルに保存します。次に例を示します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade-prep
  - du-upgrade-operator-catsrc-policy
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true
```

   b. 次のコマンドを実行して、**cgu-platform-operator-upgrade-prep.yml** ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

   c. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```

3. プラットフォーム用の **ClusterGroupUpdate** CR と、**spec.enable** フィールドを **false** に設定した Operator 更新を作成します。
   a. 次の例に示すように、ポリシーとターゲットクラスターを含むプラットフォームと Operator の更新 **ClusterGroupUpdate** CR の内容を **cgu-platform-operator-upgrade.yml** ファイルに保存します。

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
```

```
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-platform-upgrade 1
  - du-upgrade-operator-catsrc-policy 2
  - common-subscriptions-policy 3
  preCaching: true
  clusterSelector:
  - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

1. これはプラットフォーム更新ポリシーです。
2. これは、更新される Operator のカタログソース情報が含まれるポリシーです。事前キャッシュ機能がマネージドクラスターにダウンロードする Operator イメージを決定するために必要です。
3. これは、Operator を更新するためのポリシーです。

b. 次のコマンドを実行して、**cgu-platform-operator-upgrade.yml** ファイルをハブクラスターに適用します。

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. オプション: プラットフォームおよび Operator の更新用にイメージを事前キャッシュします。

a. 以下のコマンドを実行して、**ClusterGroupUpgrade** CR で事前キャッシュを有効にします。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

b. 更新プロセスを監視し、事前キャッシュが完了するまで待ちます。マネージドクラスターで次のコマンドを実行して、事前キャッシュの状態を確認します。

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

c. 以下のコマンドを実行して、更新を開始する前に事前キャッシュが完了したかどうかを確認します。

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

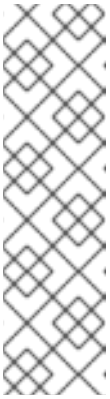
5. プラットフォームおよび Operator の更新を開始します。

a. 以下のコマンドを実行して、**cgu-du-upgrade ClusterGroupUpgrade** CR がプラットフォームと Operator の更新を開始します。

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

b. プロセスを監視します。完了したら、次のコマンドを実行して、ポリシーが準拠していることを確認します。

```
$ oc get policies --all-namespaces
```



注記

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

事前キャッシュと更新の両方で、ポリシー、配置バインディング、配置ルール、マネージドクラスターアクション、マネージドクラスタービューなどの追加リソースが作成され、手順を完了することができます。**afterCompletion.deleteObjects** フィールドを **true** に設定すると、更新の完了後にこれらのリソースがすべて削除されます。

19.10.1.5. 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

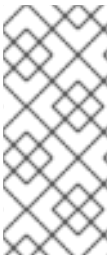
以前のバージョンの OpenShift Container Platform では、Performance Addon Operator はアプリケーションの自動低レイテンシーパフォーマンスチューニングを提供していました。OpenShift Container Platform 4.11 以降では、これらの機能は Node Tuning Operator の一部です。

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 リポジトリーを作成している。リポジトリーはハブクラスターからアクセス可能で、Argo CD のソースリポジトリーとして定義されている必要があります。
- OpenShift Container Platform 4.11 以降に更新します。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **common-ranGen.yaml** ファイル の Performance Addon Operator namespace、Operator グループ、およびサブスクリプションの **ComplianceType** を **mustnothave** に変更します。

```
- fileName: PaoSubscriptionNS.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscription.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
```

2. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。**common-subscriptions-policy** ポリシーのステータスが **Non-Compliant** に変わります。

3. Topology Aware Lifecycle Manager を使用して、ターゲットクラスターに変更を適用します。設定変更のロールアウトの詳細については、「関連情報」セクションを参照してください。

4. プロセスを監視します。ターゲットクラスターの **common-subscriptions-policy** ポリシーのステータスが **Compliant** の場合、Performance Addon Operator はクラスターから削除されています。次のコマンドを実行して、**common-subscriptions-policy** のステータスを取得します。

```
$ oc get policy -n ztp-common common-subscriptions-policy
```

5. **common-ranGen.yaml** ファイルの **.spec.sourceFiles** から Performance Addon Operator namespace、Operator グループ、およびサブスクリプション CR を削除します。

6. 変更をカスタムサイトリポジトリーにマージし、ArgoCD アプリケーションが変更をハブクラスターに同期するのを待ちます。ポリシーは準拠したままです。

19.10.2. ZTP の自動作成された ClusterGroupUpgrade CR について

TALM には、ハブ クラスター上の **ManagedCluster** CR の **Ready** 状態を監視し、ZTP (ゼロ タッチ プロビジョニング) 用の **ClusterGroupUpgrade** CR を作成する **ManagedClusterForCGU** と呼ばれるコントローラーがあります。

ztp-done ラベルが適用されていない **Ready** 状態の管理対象クラスターの場合、**ManagedClusterForCGU** コントローラーは、ZTP プロセス中に作成された関連する RHACM ポリシーを使用して、**ztp-install** namespace に **ClusterGroupUpgrade** CR を自動的に作成します。次に TALM は自動作成された **ClusterGroupUpgrade** CR にリスト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。

注記

クラスターが **Ready** になったときにマネージドクラスターにバインドされたポリシーがない場合、**ClusterGroupUpgrade** CR は作成されません。

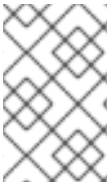
ZTP の自動作成された ClusterGroupUpgrade CR の例

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" (1)
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" (2)
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
```

1. TALM がクラスター設定を完了する際にマネージドクラスターに適用されます。
2. TALM が設定ポリシーのデプロイを開始するときにマネージドクラスターに適用されます。

19.11. GITOPS ZTP の更新

Gitops ゼロタッチプロビジョニング (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。

19.11.1. 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 ゼロタッチプロビジョニング (ZTP) を更新できます。更新プロセスにより、マネージドクラスターへの影響が回避されます。

注記

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

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

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

19.11.2. アップグレードの準備

次の手順を使用して、GitOps ゼロ タッチ プロビジョニング (ZTP) アップグレードのためにサイトを準備します。

手順

1. GitOps ZTP で使用するために Red Hat OpenShift GitOps を設定するために使用されるカスタムリソース (CR) を持つ GitOps ZTP コンテナーの最新バージョンを取得します。
2. 次のコマンドを使用して、**argocd/deployment** ディレクトリーを抽出します。

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

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

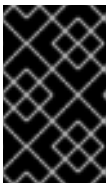
/update ディレクトリーには、次のサブディレクトリーが含まれています。

- **update/extra-manifest**: **SiteConfig** CR が追加のマニフェスト **configMap** を生成するために使用するソース CR ファイルが含まれています。
- **update/source-crs** には、**PolicyGenTemplate** CR が Red Hat Advanced Cluster Management (RHACM) ポリシーを生成するために使用するソース CR ファイルが含まれています。

- **update/argocd/deployment** には、この手順の次のステップで使用するハブクラスターに適用するパッチおよび YAML ファイルが含まれます。
- **update/argocd/example**: 推奨される設定を表す **SiteConfig** および **PolicyGenTemplate** ファイルの例が含まれています。

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

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

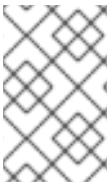


重要

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

19.11.3. 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、ゼロ タッチ プロビジョニング (ZTP) でデプロイされたマネージドクラスターをリスト表示するラベル 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、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=
```

19.11.4. 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
```

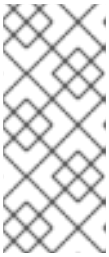
19.11.5. Git リポジトリーに必要な変更

ztp-site-generate コンテナーを以前のリリースの GitOps ZTP から v4.10 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 ゼロ タッチ プロビジョニング (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。

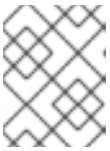
PolicyGenTemplate kustomization ファイルには、すべての **PolicyGenTemplate** YAML ファイルが **generator** セクションに含まれ、**Namespace** CR が **resource** セクションに含まれている必要があります。以下に例を示します。

```
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 コンテナーから抽出した **argocd/example** ディレクトリーにあります。

19.11.6. 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
```

19.11.7. 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。ZTP GitOps v4.10 以降 **ztp-site-generate** コンテナーでは、これらのポリシーは **inform** モードに設定され、ユーザーが追加の手順を実行しない限り、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。