OpenShift Container Platform 4.15

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: 2024-04-25

OpenShift Container Platform 4.15 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

概要

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

目次

第1章 パフォーマンスとスケーラビリティの推奨プラクティス

1.1. コントロールプレーンの推奨プラクティス

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

1.1.1. クラスターのスケーリングに関する推奨プラクティス

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

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

クラスターをノード数のより高い値にスケールアップする場合:

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

注記

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

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

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

注記

大規模で高密度のクラスターをノード数を減らしてスケールダウンする場合には、長い時間がかかる可能性があります。このプロセスで、終了するノードで実行されているオブジェクトのドレイン (解放) またはエビクトが並行して実行されるためです。また、エビクトするオブジェクトが多過ぎる場合に、クライアントはリクエストのスロットリングを開始する可能性があります。デフォルトの 1 秒あたりのクライアントクエリー数 (QPS) とバーストレートは、現在それぞれ **50** と **100** に設定されています。これらの値は、OpenShift Container Platform では変更できません。

1.1.2. コントロールプレーンノードのサイジング

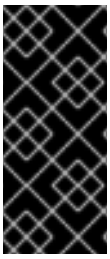
コントロールプレーンノードのリソース要件は、クラスター内のノードとオブジェクトの数とタイプによって異なります。次のコントロールプレーンノードサイズの推奨事項は、コントロールプレーン密度に焦点を当てたテストまたは **クラスター密度** の結果に基づいています。このテストでは、指定された数の namespace にわたって次のオブジェクトを作成します。

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

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

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

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



重要

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

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

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

重要

以下の設定でのみ、実行中の OpenShift Container Platform 4.15 クラスターでコントロールプレーンのノードサイズを変更できます。

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

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

重要

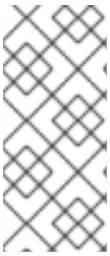
この推奨事項は、ネットワークプラグインとして OpenShift SDN を使用して OpenShift Container Platform クラスターでキャプチャーされたデータポイントに基づいています。

注記

OpenShift Container Platform 3.11 以前のバージョンと比較すると、OpenShift Container Platform 4.15 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るようになりました。サイズはこれを考慮に入れて決定されます。

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

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



注記

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

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

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

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

前提条件

- AWS クラスターは、コントロールプレーンマシンセットを使用します。

手順

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

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

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

```
providerSpec:
  value:
    ...
    instanceType: <compatible_aws_instance_type> 1
```

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

3. 変更を保存します。

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

関連情報

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

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

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

前提条件

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

手順

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

c. 停止したインスタンスを選択し、**Actions → Instance Settings → Change instance type** をクリックします。

d. インスタンスをより大きなタイプに変更し、タイプが前の選択と同じベースであることを確認して、変更を適用します。たとえば、**m6i.xlarge** を **m6i.2xlarge** または **m6i.4xlarge** に変更できます。

e. インスタンスを起動します。

f. OpenShift Container Platform クラスターにインスタンスに対応する **Machine** オブジェクトがある場合、AWS コンソールで設定されたインスタンスタイプと一致するようにオブジェクトのインスタンスタイプを更新します。

3. コントロールプレーンマシンごとにこのプロセスを繰り返します。

関連情報

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

1.2. インフラストラクチャーの推奨プラクティス

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

1.2.1. インフラストラクチャーノードのサイジング

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

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

通常、3 つのインフラストラクチャーノードはクラスターごとに推奨されます。

重要

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

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

注記

OpenShift Container Platform 3.11 以前のバージョンと比較すると、OpenShift Container Platform 4.15 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るようになりました。これは、上記のサイジングの推奨内容に影響します。

1.2.2. Cluster Monitoring Operator のスケーリング

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

1.2.3. Prometheus データベースのストレージ要件

Red Hat は、スケールサイズに応じて各種のテストを実行しました。

注記

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

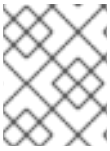
表1.1 クラスター内のノード/Pod の数に基づく Prometheus データベースのストレージ要件

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

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

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

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



注記

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

OpenShift Container Platform についての推奨事項

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

1.2.4. クラスターモニタリングの設定

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

手順

Prometheus のストレージ容量を拡張するには、以下を実行します。

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

```
apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} 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 Prometheus の保持のデフォルト値は **PROMETHEUS_RETENTION_PERIOD=15d** です。時間は、接尾辞 s、m、h、d のいずれかを使用する単位で測定されます。

2 4 クラスターのストレージクラス。

3 標準の値は **PROMETHEUS_STORAGE_SIZE=2000Gi** です。ストレージの値には、接尾辞 E、P、T、G、M、K のいずれかを使用した単純な整数または固定小数点整数を使用できます。 また、2 のべき乗の値 (Ei、Pi、Ti、Gi、Mi、Ki) を使用することもできます。

5 標準の値は **ALERTMANAGER_STORAGE_SIZE=20Gi** です。ストレージの値には、接尾辞 E、P、T、G、M、K のいずれかを使用した単純な整数または固定小数点整数を使用できます。 また、2 のべき乗の値 (Ei、Pi、Ti、Gi、Mi、Ki) を使用することもできます。

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

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

1.2.5. 関連情報

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

1.3. ETCD についての推奨されるプラクティス

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

1.3.1. etcd についての推奨されるプラクティス

etcd はデータをディスクに書き込み、プロポーザルをディスクに保持するため、そのパフォーマンスはディスクのパフォーマンスに依存します。etcd は特に I/O を集中的に使用するわけではありませんが、最適なパフォーマンスと安定性を得るには、低レイテンシーのブロックデバイスが必要です。etcd のコンセンサスプロトコルは、メタデータをログ (WAL) に永続的に保存することに依存しているため、etcd はディスク書き込みの遅延に敏感です。遅いディスクと他のプロセスからのディスクアクティビティーは、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 10 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。

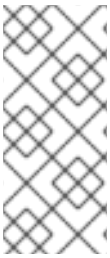
このようなパフォーマンスを実現するには、低レイテンシーで高スループットの SSD または NVMe ディスクに支えられたマシンで etcd を実行します。シングルレベルセル (SLC) ソリッドステートドライブ (SSD) を検討してください。これは、メモリーセルごとに 1 ビットを提供し、耐久性と信頼性が高く、書き込みの多いワークロードに最適です。

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。

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

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



注記

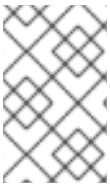
NAS または SAN のセットアップ、および回転するドライブは避けてください。Ceph Rados Block Device (RBD) およびその他のタイプのネットワーク接続ストレージでは、予測できないネットワーク遅延が発生する可能性があります。etcd ノードに大規模な高速ストレージを提供するには、PCI パススルーを使用して NVM デバイスをノードに直接渡します。

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。



注記

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

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

前提条件

- Podman や Docker などのコンテナーランタイムは、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/cloud-bulldozer/etcd-perf
```

  - Docker を使用する場合は、次のコマンドを実行します。

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

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

- **etcd_disk_wal_fsync_duration_seconds_bucket** メトリックは、etcd の WAL fsync 期間を報告します。
- **etcd_disk_backend_commit_duration_seconds_bucket** メトリクスは、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
- OpenShift Container Platform の etcd パフォーマンスに関するトラブルシューティングガイド

1.3.2. etcd を別のディスクに移動する

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

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

注記

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。
- **MachineConfigPool** は **metadata.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。

手順

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

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

```
# lsblk
```

lsblk コマンドで報告された新しいディスクのデバイス名をメモします。

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

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

```
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Make File System on /dev/<new_disk_name>
          DefaultDependencies=no
          BindsTo=dev-<new_disk_name>.device
          After=dev-<new_disk_name>.device var.mount
          Before=systemd-fsck@dev-<new_disk_name>.service

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

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

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

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

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

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

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

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

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

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

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

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

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

- 指定したディスクに XFS ファイルシステムが作成されます。
- ディスクは **/var/lib/etc** にマウントされます。
- **/sysroot/ostree/deploy/rhcos/var/lib/etcd** のコンテンツは **/var/lib/etcd** に同期されます。
- **/var/lib/etcd** の **SELinux** ラベルの復元が強制されます。
- 古いコンテンツは削除されません。

4. ノードが別のディスク上に配置されたら、次のような内容で **etcd-mc.yml** ファイルを更新し、**<new_disk_name>** のインスタンスをメモしたデバイス名に置き換えます。

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

```
    config:
      ignition:
        version: 3.4.0
      systemd:
        units:
        - contents: |
            [Unit]
            Description=Mount /dev/<new_disk_name> to /var/lib/etcd
            Before=local-fs.target
            After=var.mount

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

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

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

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

検証手順

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

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

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

出力例

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

関連情報

- Red Hat Enterprise Linux CoreOS (RHCOS)

1.3.3. etcd データのデフラグ

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

これらの主要な指標をモニターします。

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

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

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

デフラグは自動的に行われますが、手動でトリガーすることもできます。

注記

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

1.3.3.1. 自動デフラグ

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

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

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

警告

自動デフラグにより、Kubernetes コントローラ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>**

1.3.3.2. 手動デフラグ

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

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

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

警告

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

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

前提条件

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

手順

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

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

   出力例

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

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

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

出力例

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|         ENDPOINT          |        ID       | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |             91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |             91624 |        |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |             91624 |        |
+---------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

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

2. etcd メンバーのデフラグ。

   a. 実行中の etcd コンテナーに接続し、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 |   3.5.9 |  104 MB |     false |      false |
7 |      91624 |            91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.5.9 |   41 MB |     false |      false |
7 |      91624 |            91624 |        | (1)
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.5.9 |  104 MB |      true |      false |
7 |      91624 |            91624 |        |
+--------------------------+-----------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

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

e. これらの手順を繰り返して他の etcd メンバーのそれぞれに接続し、デフラグします。常に最後にリーダーをデフラグします。
etcd Pod が回復するように、デフラグアクションごとに 1 分以上待機します。etcd Pod が回復するまで、etcd メンバーは応答しません。

3. 領域のクォータの超過により **NOSPACE** アラームがトリガーされる場合、それらをクリアします。

a. **NOSPACE** アラームがあるかどうかを確認します。

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

出力例

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

b. アラームをクリアします。

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

1.3.4. etcd のチューニングパラメーターの設定

コントロールプレーンのハードウェア速度を **"Standard"**、**"Slower"**、またはデフォルトの **""** に設定できます。

デフォルト設定では、使用する速度をシステムが決定できます。システムは以前のバージョンから値を選択できるため、この値により、この機能が存在しないバージョンからのアップグレードが可能になります。

他の値のいずれかを選択すると、デフォルトが上書きされます。タイムアウトまたはハートビートの欠落が原因でリーダーの選出が多数発生し、システムが **""** または **"Standard"** に設定されている場合は、ハードウェア速度を **"Slower"** に設定して、遅延の増加に対するシステムの耐性を高めます。

重要

etcd レイテンシー許容値の調整はテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、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。

1.3.4.1. ハードウェア速度許容値の変更

etcd のハードウェア速度許容値を変更するには、次の手順を実行します。

前提条件

- クラスターインスタンスを編集してテクノロジープレビュー機能を有効にしている。詳細は、「フィーチャーゲートについて」を参照してください。

手順

1. 次のコマンドを入力して、現在の値を確認します。

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

出力例

```
Control Plane Hardware Speed:  <VALUE>
```

注記

出力が空の場合、フィールドは設定されていないため、デフォルト ("") として考慮される必要があります。

2. 次のコマンドを入力して値を変更します。**<value>** を有効な値のいずれかに置き換えます (**""**、**"Standard"**、または **"Slower"**)。

```
oc patch etcd/cluster --type=merge -p '{"spec": {"controlPlaneHardwareSpeed": "<value>"}}'
```

次の表は、各プロファイルのハートビート間隔とリーダー選出タイムアウトを示しています。これらの値は変更になる可能性があります。

プロファイル	ETCD_HEARTBEAT_INTERVAL	ETCD_LEADER_ELECTION_TIMEOUT
""	プラットフォームによって異なる	プラットフォームによって異なる
Standard (標準)	100	1000

Slower	500	2500

3. 出力を確認します。

出力例

```
etcd.operator.openshift.io/cluster patched
```

有効な値以外の値を入力すると、エラー出力が表示されます。たとえば、値 **Faster** を入力すると、出力は次のようになります。

出力例

```
The Etcd "cluster" is invalid: spec.controlPlaneHardwareSpeed: Unsupported value: "Faster": supported values: "", "Standard", "Slower"
```

4. 次のコマンドを入力して、値が変更したことを確認します。

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

出力例

```
Control Plane Hardware Speed:  ""
```

5. etcd Pod がロールアウトされるまで待ちます。

```
oc get pods -n openshift-etcd -w
```

次の出力は、master-0 の予期されるエントリーを示しています。続行する前に、すべてのマスターのステータスが **4/4 Running** になるまで待ちます。

出力例

```
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Pending             0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Pending             0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     ContainerCreating   0          0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     ContainerCreating   0          1s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         1/1     Running             0          2s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Completed           0          34s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Completed           0          36s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0         0/1     Completed           0          36s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0          0/1     Running             0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                4/4     Terminating         0          11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                0/4     Pending             0          0s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                0/4     Init:1/3            0          1s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                0/4     Init:2/3            0          2s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                0/4     PodInitializing     0          3s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0                3/4     Running             0          4s
```

```
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0        1/1     Running     0          26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0              3/4     Running     0          20s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0              4/4     Running     0          20s
```

6. 次のコマンドを入力して値を確認します。

```
$ oc describe -n openshift-etcd pod/<ETCD_PODNAME> | grep -e HEARTBEAT_INTERVAL
-e ELECTION_TIMEOUT
```

注記

これらの値はデフォルトから変更されていない可能性があります。

関連情報

フィーチャーゲートについて

第2章 オブジェクトの最大値に合わせた環境計画

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

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

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

警告

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

2.1. 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

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。

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

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

最大値のタイプ	4.x テスト済みの最大値
namespace の数[5]	10,000
ビルド数	10,000(デフォルト Pod RAM 512 Mi)- Source-to-Image (S2I) ビルドストラテジー
namespace ごとの Pod の数[6]	25,000
Ingress Controller ごとのルートとバックエンドの数	ルーターあたり 2,000
シークレットの数	80,000
config map の数	90,000
サービスの数[7]	10,000
namespace ごとのサービス数	5,000
サービスごとのバックエンド数	5,000
namespace ごとのデプロイメントの数[6]	2,000
ビルド設定の数	12,000
カスタムリソース定義 (CRD) の数	1,024 [8]

1. 一時停止 Pod は、2000 ノードスケールで OpenShift Container Platform のコントロールプレーンコンポーネントにストレスをかけるためにデプロイされました。同様の数値にスケーリングできるかどうかは、特定のデプロイメントとワークロードのパラメーターによって異なります。

2. ここで表示される Pod 数はテスト用の Pod 数です。実際の Pod 数は、アプリケーションのメモリー、CPU、およびストレージ要件により異なります。

3. これは、31 台のサーバー (3 つのコントロールプレーン、2 つのインフラストラクチャーノード、および 26 のワーカーノード) を備えたクラスターでテストされました。2,500 のユーザー Pod が必要な場合は、各ノードが 2000 超の Pod を内包できる規模のネットワークを割り当てるために **hostPrefix** を **20** に設定し、カスタム kubelet 設定で **maxPods** を **2500** に設定する必要があります。詳細は、OCP 4.13 でノードごとに 2500 Pod を実行する を参照してください。

4. **OVNKubernetes** ネットワークプラグインを使用するクラスターの場合、ノードごとにテストされる最大 Pod 数は 2,500 です。**OpenShiftSDN** ネットワークプラグインのノードごとにテストされる最大 Pod 数は 500 Pod です。

5. 有効なプロジェクトが多数ある場合、キースペースが過剰に拡大し、スペースのクォータを超過すると、etcd はパフォーマンスの低下による影響を受ける可能性があります。etcd ストレージを解放するために、デフラグを含む etcd の定期的なメンテナンスを行うことを強く推奨しま

す。

6. システムには、状態遷移への対応として、指定された namespace 内のすべてのオブジェクトに対して反復処理する必要がある制御ループがいくつかあります。単一の namespace に特定タイプのオブジェクトの数が多くなると、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。

7. 各サービスポートと各サービスのバックエンドには、**iptables** に対応するエントリーがあります。特定のサービスのバックエンドの数は、**Endpoints** オブジェクトのサイズに影響を与え、システム全体に送信されるデータのサイズに影響を与えます。

8. 29 台のサーバーでテストされたクラスター：3 つのコントロールプレーン、2 つのインフラストラクチャーノード、および 24 ワーカーノード。クラスターには 500 の namespace がありました。OpenShift Container Platform では、OpenShift Container Platform によってインストールされるカスタムリソース定義 (CRD)、OpenShift Container Platform と統合される製品、およびユーザーが作成した CRD を含むカスタムリソース定義 (CRD) の合計数が 1,024 に制限されます。作成された CRD の数が 1,024 を超える場合、**oc** コマンドリクエストのスロットリングが適用される可能性があります。

2.1.1. シナリオ例

例として、OpenShift Container Platform 4.15、OVN-Kubernetes ネットワークプラグイン、および以下のワークロードオブジェクトを使用して、500 個のワーカーノード (m5.2xl) がテストされ、サポートされています。

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

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

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

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

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

2.2.1. AWS クラウドプラットフォーム:

ノード	フレーバー	vCPU	RAM(GiB)	ディスクタイプ	ディスクサイズ(GiB)/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。

2.2.2. IBM Power プラットフォーム

ノード	vCPU	RAM(GiB)	ディスクタイプ	ディスクサイズ (GiB)/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。

2.2.3. IBM Z プラットフォーム

ノード	vCPU [4]	RAM(GiB)[5]	ディスクタイプ	ディスクサイズ (GiB)/IOS	カウント
コントロールプレーン/etcd [1,2]	8	32	ds8k	300 / LCU 1	3
コンピュート [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。

2.3. テスト済みのクラスターの最大値に基づく環境計画

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

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

シナリオ例

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

```
2200 / 500 = 4.4
```

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

```
2200 / 20 = 110
```

ここでは、以下のようになります。

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

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

2.4. アプリケーション要件に合わせて環境計画を立てる方法

アプリケーション環境の例を考えてみましょう。

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

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

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

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

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

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

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

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

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

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

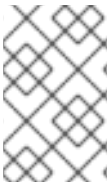
引数の長さが許可される値を超え、サービス名の文字数がこれに影響する場合、namespace の Pod は起動に失敗し始めます。たとえば、5000 サービスを含む namespace では、サービス名の制限は 33 文字であり、これにより namespace で 5000 Pod を実行できます。

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

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

注記

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



注記

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

3.1. CPU のオーバーコミットの管理

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

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

- LPAR レベル (PR/SM ハイパーバイザー) で、利用可能な物理コア (IFL) を各 LPAR に割り当てないようにします。たとえば、4 つの物理 IFL が利用可能な場合は、それぞれ 4 つの論理 IFL を持つ 3 つの LPAR を定義しないでください。
- LPAR 共有および重みを確認します。
- 仮想 CPU の数が多すぎると、パフォーマンスに悪影響を与える可能性があります。論理プロセッサーが 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

3.2. TRANSPARENT HUGE PAGES (THP) の無効

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

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

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

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

手順

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

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

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

3.4. ネットワーク設定の選択

ネットワークスタックは、OpenShift Container Platform などの Kubernetes ベースの製品の最も重要なコンポーネントの 1 つです。IBM Z® セットアップでは、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

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

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

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

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

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

手順

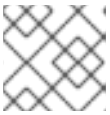
1. 以下の MCO サンプルプロファイルをコントロールプレーンノードの YAML ファイルにコピーします。たとえば、**05-master-kernelarg-hpav.yaml** です。

```
$ 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

3.6. IBM Z ホストの RHEL KVM の推奨事項

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

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

3.6.1. 仮想ブロックデバイスの I/O スレッドの使用

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

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

I/O スレッド仕様のサンプル

```
...
<domain>
  <iothreads>3</iothreads> 1
    ...
     <devices>
       ...
         <disk type="block" device="disk"> 2
<driver ... iothread="2"/>
    </disk>
       ...
     </devices>
    ...
</domain>
```

1 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。

3.6.2. 仮想 SCSI デバイスの回避

SCSI 固有のインターフェイスを介してデバイスに対応する必要がある場合にのみ、仮想 SCSI デバイスを設定します。ホスト上でバッ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。

3.6.3. ディスクについてのゲストキャッシュの設定

ホストではなく、ゲストでキャッシュするようにディスクデバイスを設定します。

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

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

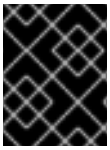
3.6.4. メモリーバルーンデバイスを除外します。

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

- アクティブなプロファイルのリストを確認します。

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

3.6.5. ホストスケジューラーの CPU 移行アルゴリズムの調整



重要

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

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

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

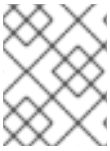
間隔を 60000 ns に動的に設定するには、以下のコマンドを入力します。

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

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

```
kernel.sched_migration_cost_ns=60000
```

3.6.6. cpuset cgroup コントローラーの無効化



注記

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

手順

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

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

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

    a. すべての仮想マシンを停止します。

    b. 以下のコマンドを実行します。

    ```
    # systemctl restart libvirtd
    ```

    c. 仮想マシンを再起動します。

この設定は、ホストの再起動後も維持されます。

3.6.7. アイドル状態の仮想 CPU のポーリング期間の調整

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

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

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

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

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

関連情報

- Linux on IBM Z® Performance Tuning for KVM
- IBM Z® での仮想化の使用

第4章 NODE TUNING OPERATOR の使用

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

4.1. NODE TUNING OPERATOR について

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

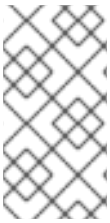
Operator は、コンテナー化された OpenShift Container Platform の TuneD デーモンを Kubernetes デーモンセットとして管理します。これにより、カスタムチューニング仕様が、デーモンが認識する形式でクラスターで実行されるすべてのコンテナー化された TuneD デーモンに渡されます。デーモンは、ノードごとに 1 つずつ、クラスターのすべてのノードで実行されます。

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、パフォーマンスプロファイルコントローラーを使用して自動チューニングを実装し、OpenShift Container Platform アプリケーションの低レイテンシーパフォーマンスを実現します。

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

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



注記

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

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

注記

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

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

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

手順

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

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

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

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 の将来のバージョンで非推奨になる予定です。

4.3. クラスターに設定されるデフォルトのプロファイル

以下は、クラスターに設定されるデフォルトのプロファイルです。

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

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

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

4.4. TUNED プロファイルが適用されていることの確認

クラスターノードに適用されている Tune D プロファイルを確認します。

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

出力例

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

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

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

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

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

出力例

```
NAME          VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE   MESSAGE
node-tuning   4.15.1    True        False         True       60m     1/5 Profiles with bootcmdline conflict
```

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

4.5. カスタムチューニング仕様

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

複数のカスタムチューニング仕様は、Operator の namespace に複数の CR として共存できます。新規 CR の存在または古い CR の削除は Operator によって検出されます。既存のカスタムチューニング仕様はすべてマージされ、コンテナー化された 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)
    tunedConfig:
      reapply_sysctl: <bool> (9)
```

(1) オプション:

(2) キー/値の **MachineConfig** ラベルのディクショナリー。キーは一意である必要があります。

(3) 省略する場合は、優先度の高いプロファイルが最初に一致するか、**machineConfigLabels** が設定されていない限り、プロファイルの一致が想定されます。

(4) オプションのリスト。

(5) プロファイルの順序付けの優先度。数値が小さいほど優先度が高くなります (**0** が最も高い優先度になります)。

(6) 一致に適用する TuneD プロファイル。例: **tuned_profile_1**

(7) オプションのオペランド設定。

(8) TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は **true**、オフの場合は **false** です。デフォルトは **false** です。

(9) TuneD デーモンの **reapply_sysctl** 機能をオンまたはオフにします。オプションは on で **true**、オフの場合は **false** です。

<match> は、以下のように再帰的に定義されるオプションの一覧です。

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

(1) ノードまたは Pod のラベル名。

(2) オプションのノードまたは Pod のラベルの値。省略されている場合も、**<label_name>** があるだけで一致条件を満たします。

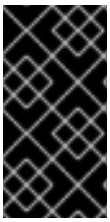
(3) オプションのオブジェクトタイプ (**node** または **pod**)。省略されている場合は、**node** が想定されます。

(4) オプションの **<match>** リスト。

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

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

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



重要

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

例: ノードまたは Pod のラベルベースのマッチング

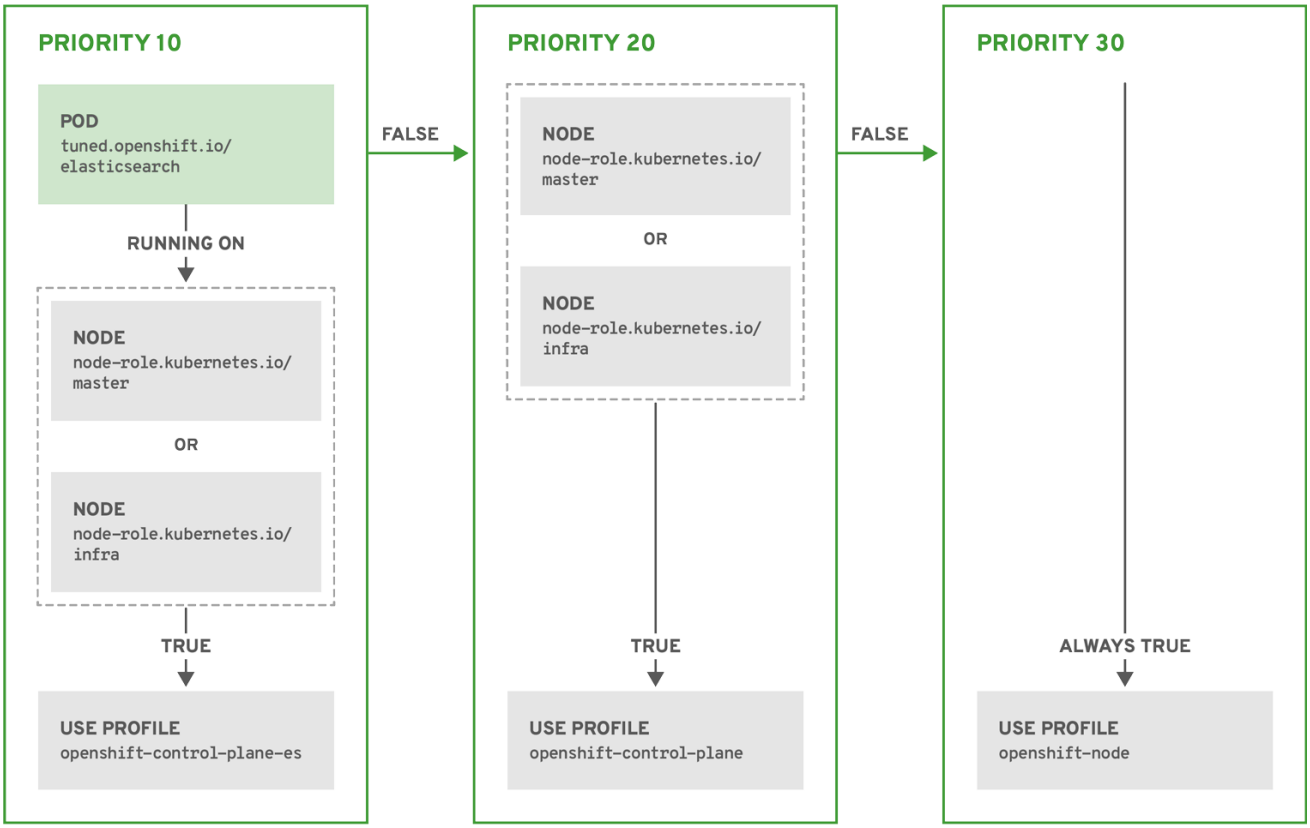
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra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



OPENSHIFT_10_0319

例: マシン設定プールベースのマッチング

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

  recommend:
  - machineConfigLabels:
```

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

ノードの再起動を最小限にするには、ターゲットノードにマシン設定プールのノードセ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。

クラウドプロバイダー固有の TuneD プロファイル

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

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

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

GCE クラウドプロバイダープロファイルの例

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

注記

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

4.6. カスタムチューニングの例

デフォルト CR からの TuneD プロファイルの使用

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

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

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

重要

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

ビルトイン TuneD プロファイルの使用

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

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

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

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

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

  recommend:
```

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

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

ホストレベルの sysctl のオーバーライド

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

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

例: ホストレベルの sysctl のオーバーライド

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

4.7. サポートされている TUNED デーモンプラグイン

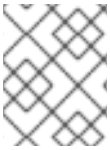
[main] セクションを除き、以下の TuneD プラグインは、Tuned CR の **profile:** セクションで定義されたカスタムプロファイルを使用する場合にサポートされます。

- audio
- cpu

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

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

- script
- systemd



注記

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

関連情報

- 利用可能な TuneD プラグイン
- TuneD を使い始める

4.8. ホステッドクラスターにおけるノードのチューニング設定

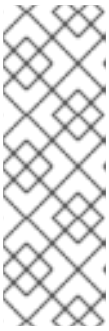
ホストされたクラスター内のノードでノードレベルのチューニングを設定するには、Node Tuning Operator を使用できます。ホストされたコントロールプレーンでは、**Tuned** オブジェクトを含む設定マップを作成し、ノードプールでそれらの設定マップを参照することで、ノードのチューニングを設定できます。

手順

1. チューニングされた有効なマニフェストを含む設定マップを作成し、ノードプールでマニフェ

ストを参照します。次の例で **Tuned** マニフェストは、任意の値を持つ **tuned-1-node-label** ノードラベルを含むノード上で **vm.dirty_ratio** を 55 に設定するプロファイルを定義します。次の **ConfigMap** マニフェストを **tuned-1.yaml** という名前のファイルに保存します。

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



注記

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

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

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

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

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

```
spec:
  ...
  tuningConfig:
  - name: tuned-1
status:
...
```

注記

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

検証

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

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

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

出力例

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

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

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

出力例

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

注記

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

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

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

出力例

```
vm.dirty_ratio = 55
```

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

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

手順

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

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

注記

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

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

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

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

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

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

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

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

注記

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

5. 管理クラスターに **NodePool** を作成します。

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

検証

ノードが使用可能になると、コンテナー化された TuneD デーモンが、適用された Tuned プロファイルに基づいて、必要なカーネルブートパラメーターを計算します。ノードの準備が整い、一度再起動して

生成された **MachineConfig** オブジェクトを適用したら、TuneD プロファイルが適用され、カーネルブートパラメーターが設定されていることを確認できます。

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

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

出力例

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

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

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

出力例

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

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

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

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

出力例

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

関連情報

ホストされたコントロールプレーンの詳細は、ホストされたコントロールプレーン を参照してください。

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

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

CPU マネージャーは、以下のような属性が含まれるワークロードに有用です。

- できるだけ長い CPU 時間が必要な場合
- プロセッサ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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause:3.2
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  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 Device Manager などの Hint Provider からトポロジーのヒントを収集し、収集したヒントを使用して **Pod** リソースを調整することで、すべての QoS (Quality of Service) クラスの **Pod** リソースを調整します。

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

none ポリシー

これはデフォルトのポリシーで、トポロジーの配置は実行しません。

best-effort ポリシー

best-effort トポロジー管理ポリシーを持つ Pod のそれぞれのコンテナーの場合、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

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

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

```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
```

```
cpuManagerPolicy: static 1
cpuManagerReconcilePeriod: 5s
topologyManagerPolicy: single-numa-node 2
```

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

5.4. POD の TOPOLOGY MANAGER ポリシーとの対話

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

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

```
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 Device Manager) を参照して、Pod のトポロジーヒントを取得します。

Topology Manager はこの情報を使用して、このコンテナ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。

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。

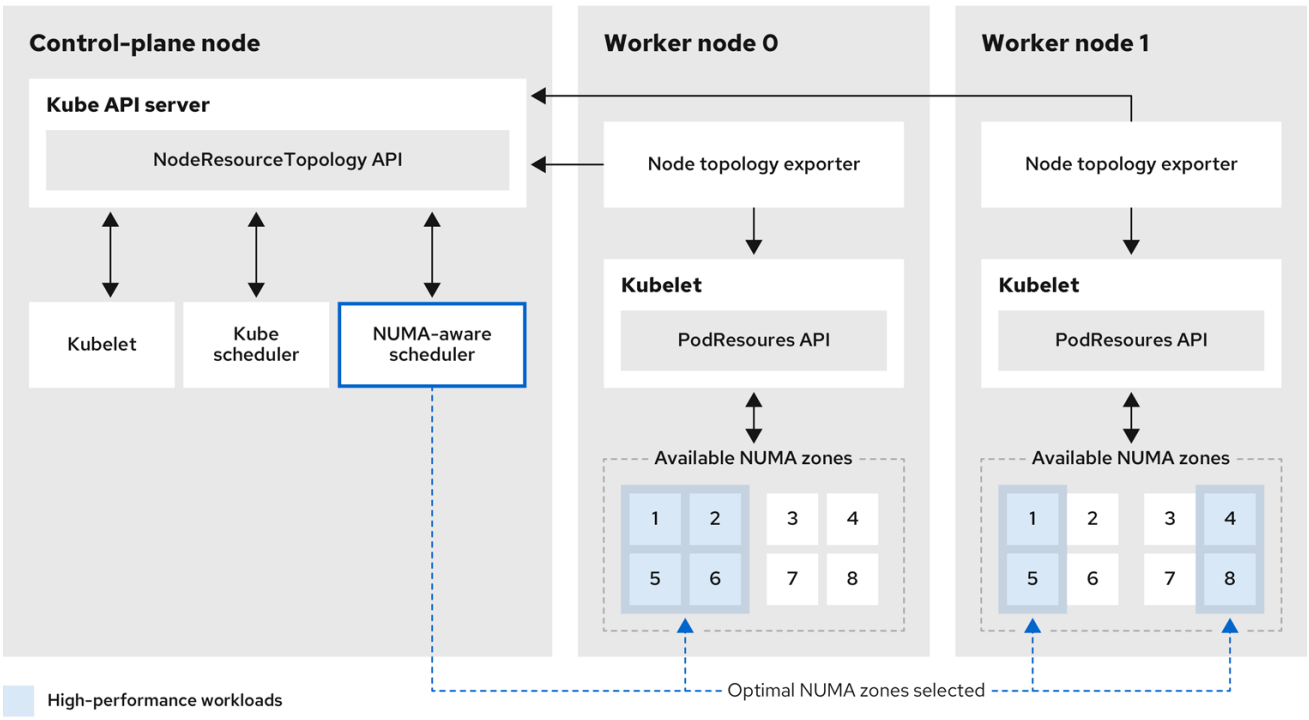
NUMA アーキテクチャーにより、複数のメモリーコントローラーを備えた CPU は、メモリーが配置されている場所に関係なく、CPU コンプレックス全体で使用可能なメモリーを使用できます。これにより、パフォーマンスを犠牲にして柔軟性を高めることができます。NUMA ゾーン外のメモリーを使用してワークロードを処理する CPU は、単一の NUMA ゾーンで処理されるワークロードよりも遅くなります。また、I/O に制約のあるワークロードの場合、離れた NUMA ゾーンのネットワークインターフェイスにより、情報がアプリケーションに到達する速度が低下します。通信ワークロードなどの高性能ワークロードは、こ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。

Node Tuning Operator のパフォーマンスプロファイルを NUMA 対応スケジューリングと統合することで、CPU アフィニティーをさらに設定し、レイテンシーの影響を受けやすいワークロードのパフォーマンスを最適化できます。

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

注記

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

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

関連情報

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

6.2. NUMA RESOURCES OPERATOR のインストール

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

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

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

前提条件

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

手順

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

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

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

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

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

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

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

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

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

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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: numaresources-operator
  namespace: openshift-numaresources
spec:
  channel: "4.15"
  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.v4.15.2   numaresources-operator   4.15.2              Succeeded
```

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

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

手順

1. NUMA Resources Operator の namespace を作成します。

   a. OpenShift Container Platform Web コンソールで、**Administration → Namespaces** をクリックします。

   b. **Create Namespace**をクリックし、**Name** フィールドに **openshift-numresources** と入力して **Create** をクリックします。

2. NUMA Resources Operator をインストールします。

   a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。

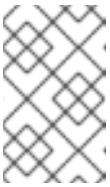
   b. 利用可能な Operator のリストから **NUMA Resources Operator** を選択し、**Install** をクリックします。

   c. **Installed Namespaces** フィールドで、**openshift-umaresources** namespace を選択して **Install** をクリックします。

3. オプション: NUMA Resources Operator が正常にインストールされたことを確認します。

   a. **Operators → Installed Operators** ページに切り替えます。

   b. **NUMA Resources Operator** が **openshift-umaresources** namespace にリストされ、**Status** が **InstallSucceeded** であることを確認します。



注記

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

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

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

6.3. NUMA 対応ワークロードのスケジューリング

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

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

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

前提条件

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

手順

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

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

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

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

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

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

検証

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

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

出力例

```
NAME                 AGE
numaresourcesoperator   10m
```

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

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

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- 必要なマシン設定プールを作成している。
- NUMA Resources Operator をインストールしている。

手順

1. **PerformanceProfile** カスタムリソース (CR) を作成します。

   a. 次の YAML を **nro-perfprof.yaml** ファイルに保存します。

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

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

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

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

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

出力例

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

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

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

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-rhel9:v4.15"
  cacheResyncPeriod: "5s" 1
```

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

注記

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

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

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

検証

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

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

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

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

出力例

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

関連情報

- Performance Profile Creator の概要。

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

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

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

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

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

出力例

```
topo-aware-scheduler
```

2. **topo-aware-scheduler** という名前のスケジューラーを使用する **Deployment** CR を作成します。次に例を示します。

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: numa-deployment-1
  namespace: openshift-numaresources
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      schedulerName: topo-aware-scheduler (1)
      containers:
      - name: 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一覧表示されていることを確認します。

a. 次のコマンドを実行して、デプロイメント Pod を実行しているノードを特定します。このとき、<namespace> は **Deployment** CR で指定した namespace に置き換えます。

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

出力例

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

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

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

出力例

```
...

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

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

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

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

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

出力例

```
Guaranteed
```

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

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

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

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

前提条件

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

手順

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

注記

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

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

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

```
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

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

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

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

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

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

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

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

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

検証

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

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

出力例

```
NAME                     AGE
numaresourcesoperator   10m
```

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

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

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

前提条件

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

手順

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cnf-worker-tuning
spec:
  machineConfigPoolSelector:
    matchLabels:
      cnf-worker-tuning: enabled
  kubeletConfig:
    cpuManagerPolicy: "static" 1
    cpuManagerReconcilePeriod: "5s"
    reservedSystemCPUs: "0,1"
    memoryManagerPolicy: "Static" 2
    evictionHard:
      memory.available: "100Mi"
    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/v1
```

```
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.15"
  cacheResyncPeriod: "5s" (1)
```

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

注記

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

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

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

検証

- 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
```

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

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

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

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

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

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

出力例

```
topo-aware-scheduler
```

2. **topo-aware-scheduler** という名前のスケジューラーを使用する **Deployment** CR を作成します。次に例を示します。

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: numa-deployment-1
  namespace: openshift-numaresources
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      labels:
        app: test
    spec:
      schedulerName: topo-aware-scheduler (1)
      containers:
      - name: 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一覧表示されていることを確認します。

   a. 次のコマンドを実行して、デプロイメント Pod を実行しているノードを特定します。このとき、<namespace> は **Deployment** CR で指定した namespace に置き換えます。

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

   出力例

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

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

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

   出力例

   ```
   ...

   Zones:
     Costs:
       Name:   node-0
       Value:  10
       Name:   node-1
       Value:  21
     Name:     node-0
     Resources:
       Allocatable:  39
       Available:    21 (1)
       Capacity:     40
       Name:         cpu
       Allocatable:  6442450944
       Available:    6442450944
       Capacity:     6442450944
       Name:         hugepages-1Gi
       Allocatable:  134217728
       Available:    134217728
       Capacity:     134217728
       Name:         hugepages-2Mi
       Allocatable:  262415904768
       Available:    262206189568
   ```

```
  Capacity:    270146007040
  Name:        memory
Type:          Node
```

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

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

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

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

出力例

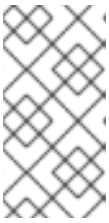
```
Guaranteed
```

6.5. オプション: NUMA リソース更新のポーリング操作の設定

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

設定オプションは次のとおりです。

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



注記

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

前提条件

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

- **cluster-admin** 権限を持つユーザーとしてログインしている。
- NUMA Resources Operator をインストールしている。

手順

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

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

(1) 有効な値は **Periodic**、**Events**、**PeriodicAndEvents** です。**Periodic** を使用して、**infoRefreshPeriod** で定義した間隔で kubelet をポーリングします。**Events** を使用して、Pod のライフサイクルイベントごとに kubelet をポーリングします。両方のメソッドを有効にするには、**PeriodicAndEvents** を使用します。

(2) **Periodic** または **PeriodicAndEvents** リフレッシュモードのポーリング間隔を定義します。リフレッシュモードが **Events** の場合、このフィールドは無視されます。

(3) 有効な値は **Enabled** と **Disabled** です。**NUMAResourcesScheduler** の **cacheResyncPeriod** 仕様では、**Enabled** への設定が必須です。

検証

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

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

出力例

```
...

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

...
```

6.6. NUMA 対応スケジューリングのトラブルシューティング

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

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。
- cluster-admin 権限を持つユーザーとしてログインしている。
- NUMA Resources Operator をインストールし、NUMA 対応のセカンダリースケジューラーをデプロイします。

手順

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

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

出力例

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

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

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

出力例

```
topo-aware-scheduler
```

3. NUMA 対応のスケジュール可能なノードに **noderesourcetopologies** CR が適用されていることを確認します。以下のコマンドを実行します。

```
$ oc get noderesourcetopologies.topology.node.k8s.io
```

出力例

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

注記

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

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/v1
  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/v1
  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/v1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.15"
  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。

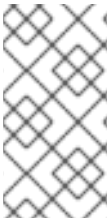
```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

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
```

6.6.4. NUMA Resources Operator データの収集

oc adm must-gather CLI コマンドを使用すると、NUMA Resources Operator に関連付けられた機能やオブジェクトなど、クラスターに関する情報を収集できます。

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

手順

- **must-gather** を使用して NUMA Resources Operator データを収集するには、NUMA Resources Operator の **must-gather** イメージを指定する必要があります。

```
$ oc adm must-gather --image=registry.redhat.io/numaresources-must-gather/numaresources-must-gather-rhel9:4.15
```

第7章 スケーラビリティとパフォーマンスの最適化

7.1. ストレージの最適化

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

7.1.1. 利用可能な永続ストレージオプション

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

表7.1 利用可能なストレージオプション

ストレージタイプ	説明	例
ブロック	● 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。

7.1.2. 設定可能な推奨のストレージ技術

以下の表では、特定の OpenShift Container Platform クラスタ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7.2 設定可能な推奨ストレージ技術

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

[1] ReadOnlyMany

[2] ReadWriteMany

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

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

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

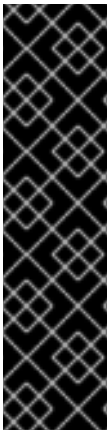
[6] ログについては、ログストアの永続ストレージの設定セクションで推奨されるストレージソリューションを確認してください。NFS ストレージを永続ボリュームとして使用するか、Gluster などの NAS を介して使用すると、データが破損する可能性があります。したがって、NFS は、OpenShift Container Platform Logging の Elasticsearch ストレージおよび LokiStack ログストアではサポートされていません。ログストアごとに 1 つの永続的なボリュームタイプを使用する必要があります。

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

注記

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

7.1.2.1. 特定アプリケーションのストレージの推奨事項



重要

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

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

7.1.2.1.1. レジストリー

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

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

7.1.2.1.2. スケーリングされたレジストリー

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

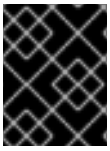
- ストレージ技術は、RWX アクセスモードをサポートする必要があります。
- ストレージ技術は、リードアフターライト (Read-After-Write) の一貫性を確保する必要があります。
- 推奨されるストレージ技術はオブジェクトストレージです。
- Red Hat OpenShift Data Foundation (ODF)、Amazon Simple Storage Service (Amazon S3)、Google Cloud Storage (GCS)、Microsoft Azure Blob Storage、および OpenStack Swift がサポートされています。
-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

- ブロックストレージは設定できません。

7.1.2.1.3. メトリクス

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

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



重要

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

7.1.2.1.4. ロギング

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

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

注記

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

7.1.2.1.5. アプリケーション

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

- 動的な PV プロビジョニングをサポートするストレージ技術は、マウント時のレイテンシーが低く、ノードに関連付けられておらず、正常なクラスターをサポートします。
- アプリケーション開発者はアプリケーションのストレージ要件や、それがどのように提供されているストレージと共に機能するかを理解し、アプリケーションのスケー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。

7.1.2.2. 特定のアプリケーションおよびストレージの他の推奨事項

重要

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

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder は ROX アクセスモードのユー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。

7.1.3. データストレージ管理

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

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

ディレクトリー	注記	サイジング	予想される拡張
/var/lib/etcd	データベースを保存する際に etcd ストレージに使用されます。	20 GB 未満。 データベースは、最大 8 GB まで拡張できます。	環境と共に徐々に拡張します。メタデータのみを格納します。 メモリーに 8 GB が追加されるたびに 20-25 GB を追加します。
/var/lib/containers	これは CRI-O ランタイムのマウントポイントです。アクティブなコンテナーランタイム (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/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。**

ディレクトリー	注記	サイジング	予想される拡張
/var/lib/kubelet	Pod の一時ボリュームストレージです。これには、ランタイムにコンテナ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。**

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

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

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

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

7.1.5. 関連情報

- Elasticsearch ログストアの設定

7.2. ルーティングの最適化

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

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

OpenShift Container Platform Ingress コントローラー (ルーター) は、ルートとイン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
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 シャーディングの詳細は、ルートラベルを使用した Ingress コントローラーのシャーディング設定 および namespace ラベルを使用した Ingress コントローラーのシャーディング設定 を参照してください。

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

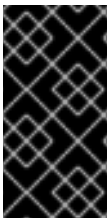
7.2.2. Ingress コントローラー (ルーター) liveness、readiness、および startup プローブの設定

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

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

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

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



重要

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

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

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

検証

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

7.2.3. HAProxy リロード間隔の設定

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

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

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

警告

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

手順

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

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

7.3. ネットワークの最適化

OpenShift SDN は OpenvSwitch、VXLAN (Virtual extensible LAN) トンネル、OpenFlow ルール、iptables を使用します。このネットワークは、ジャンボフレーム、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、Generic Network Virtualization Encapsulation (Geneve)を使用します。このネットワークは、ネットワークインターフェイスコントローラー(NIC)のオフロードを使用して調整できます。

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。

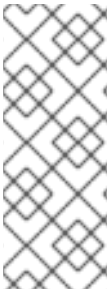
VXLAN オフロードはレイテンシーを短縮しません。ただし、CPU の使用率はレイテンシーテストでも削減されます。

7.3.1. ネットワークでの MTU の最適化

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

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

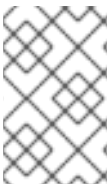
OpenShift SDN ネットワークプラグインオーバーレイ MTU は、NIC MTU よりも少なくとも 50 バイト小さくする必要があります。これは、SDN オーバーレイのヘッダーに相当します。したがって、通常のイーサネットネットワークでは、これを **1450** に設定する必要があります。ジャンボフレームイーサネットネットワークでは、これを **8950** に設定する必要があります。これらの値は、NIC に設定された MTU に基づいて、Cluster Network Operator によって自動的に設定される必要があります。したがって、クラスター管理者は通常、これらの値を更新しません。Amazon Web Services (AWS) およびベアメタル環境は、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。



注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

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



注記

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

7.3.2. 大規模なクラスターのインストールに推奨されるプラクティス

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

```
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
```

```
- cidr: 10.0.0.0/16
networkType: OVNKubernetes
serviceNetwork:
- 172.30.0.0/16
```

クラスターのサイズが 500 を超える場合、デフォルトのクラスターネットワーク **cidr 10.128.0.0/14** を使用することはできません。500 ノードを超えるノード数にするには、**10.128.0.0/12** または **10.128.0.0/10** に設定する必要があります。

7.3.3. IPsec の影響

ノードホストの暗号化、復号化に CPU 機能が使用されるので、使用する IP セキュリティ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。

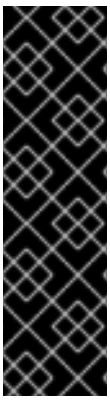
IPSec は、NIC に到達する前に IP ペイロードレベルでトラフィックを暗号化して、NIC オフロードに使用されてしまう可能性のあるフィールドを保護します。つまり、IPSec が有効な場合には、NIC アクセラレーション機能を使用できない場合があり、スループットの減少、CPU 使用率の上昇につながります。

7.3.4. 関連情報

- 高度なネットワーク設定パラメーターの変更
- OVN-Kubernetes ネットワークプラグインの設定パラメーター
- OpenShift SDN ネットワークプラグインの設定パラメーター
- ワーカーレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスターの安定性の向上

7.4. マウント NAMESPACE のカプセル化による CPU 使用率の最適化

マウント namespace のカプセル化を使用して kubelet および CRI-O プロセスにプライベート namespace を提供することで、OpenShift Container Platform クラスターでの CPU 使用率を最適化できます。これにより、機能に違いはなく、systemd が使用するクラスター CPU リソースが削減されます。



重要

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

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

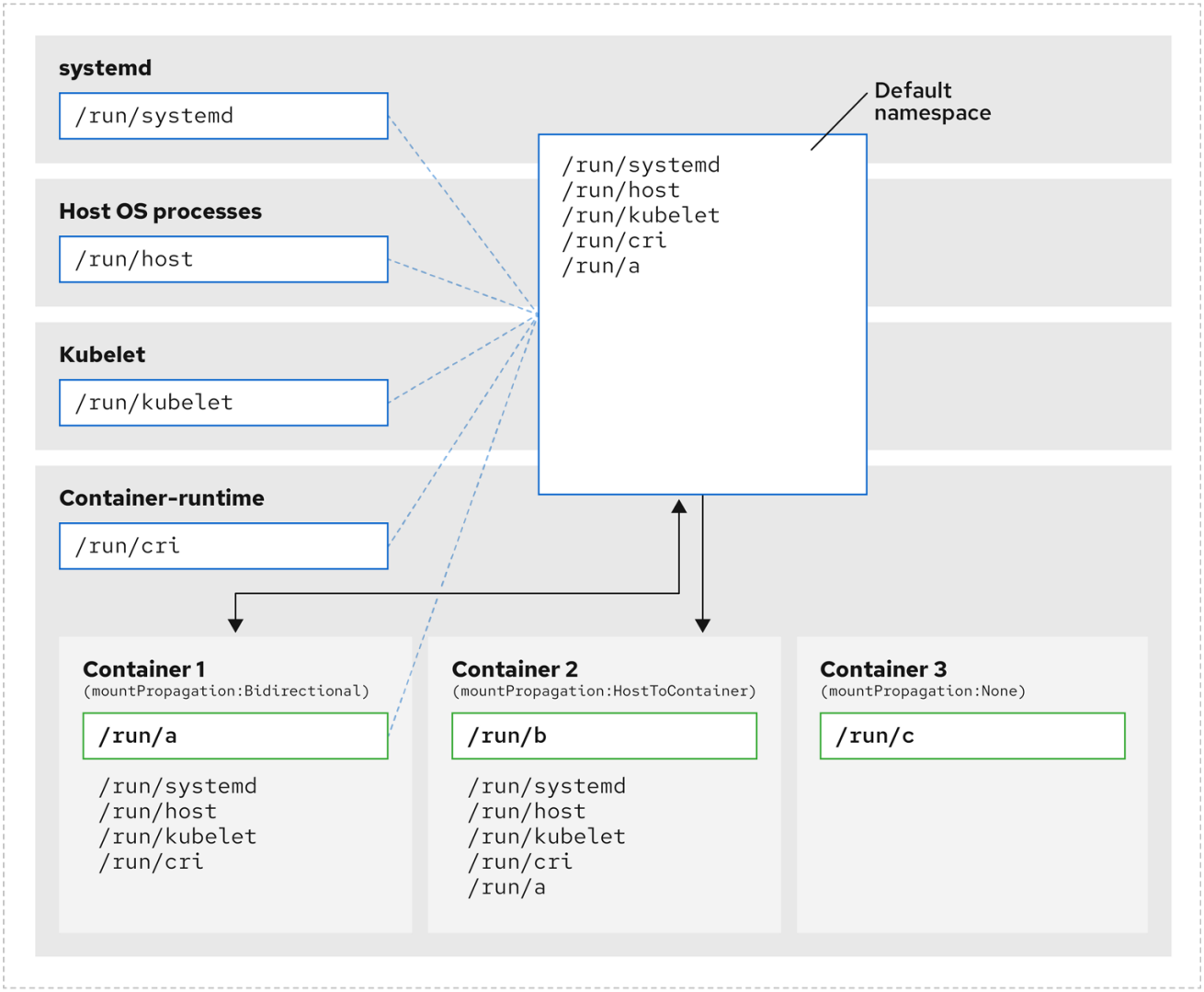
7.4.1. マウント namespace のカプセル化

マウント namespace は、異なる namespace のプロセスが互いのファイルを表示できないように、マウントポイントを分離するために使用されます。カプセル化は、Kubernetes マウント namespace を、ホストオペレーティングシステムによって常にスキャンされない別の場所に移動するプロセスです。

ホストオペレーティングシステムは systemd を使用して、すべてのマウント namespace (標準の Linux マウントと、Kubernetes が操作に使用する多数のマウントの両方) を常にスキャンします。kubelet と CRI-O の現在の実装はどちらも、すべてのコンテナーランタイムと kubelet マウントポイントに最上位の namespace を使用します。ただし、これらのコンテナー固有のマウントポイントをプライベート namespace にカプセル化すると、systemd のオーバーヘッドが削減され、機能に違いはありません。CRI-O と kubelet の両方に個別のマウント namespace を使用すると、systemd または他のホスト OS の相互作用からコンテナー固有のマウントをカプセル化できます。

CPU の大幅な最適化を潜在的に達成するこの機能は、すべての OpenShift Container Platform 管理者が利用できるようになりました。カプセル化は、Kubernetes 固有のマウントポイントを特権のないユーザーによる検査から安全な場所に保存することで、セキュリティーを向上させることもできます。

次の図は、カプセル化の前後の Kubernetes インストールを示しています。どちらのシナリオも、双方向、ホストからコンテナー、およびなしのマウント伝搬設定を持つコンテナーの例を示しています。



290_OpenShift_1122

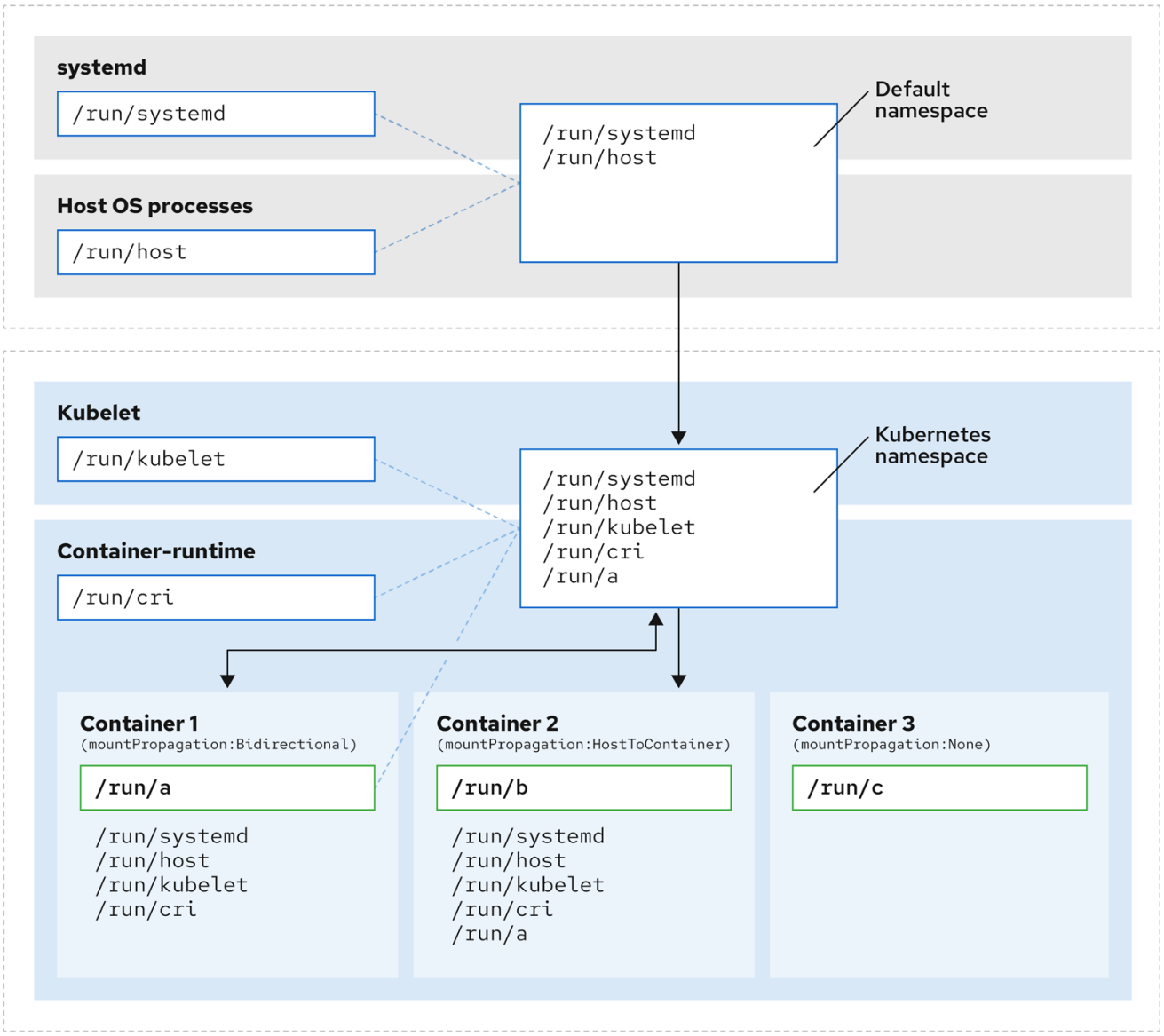
ここでは、systemd、ホストオペレーティングシステムプロセス、kubelet、およびコンテナーランタイムが単一のマウント namespace を共有していることがわかります。

- systemd、ホストオペレーティングシステムプロセス、kubelet、およびコンテナーランタイムはそれぞれ、すべてのマウントポイントにアクセスして可視化できます。
- コンテナー 1 は、双方向のマウント伝達で設定され、systemd およびホストマウント、kubelet および CRI-O マウントにアクセスできます。**/run/a** などのコンテナー 1 で開始されたマウント

は、systemd、ホスト OS プロセス、kubelet、コンテナーランタイム、およびホストからコンテナーへのまたは双方向のマウント伝達が設定されている他のコンテナー (コンテナー 2 のように) に表示されます。

- ホストからコンテナーへのマウント伝達で設定されたコンテナー 2 は、systemd およびホストマウント、kubelet および CRI-O マウントにアクセスできます。**/run/b** などのコンテナー 2 で発生したマウントは、他のコンテキストからは見えません。
- マウント伝達なしで設定されたコンテナー 3 には、外部マウントポイントが表示されません。**/run/c** などのコンテナー 3 で開始されたマウントは、他のコンテキストからは見えません。

次の図は、カプセル化後のシステム状態を示しています。



290_OpenShift_1122

- メインの systemd プロセスは、Kubernetes 固有のマウントポイントの不要なスキャンに専念しなくなりました。systemd 固有のホストマウントポイントのみを監視します。
- ホストオペレーティングシステムプロセスは、systemd およびホストマウントポイントにのみアクセスできます。

- CRI-O と kubelet の両方に個別のマウント namespace を使用すると、すべてのコンテナー固有のマウントが systemd または他のホスト OS の対話から完全に分離されます。
- コンテナー 1 の動作は変更されていませんが、**/run/a** などのコンテナーが作成するマウントが systemd またはホスト OS プロセスから認識されなくなります。kubelet、CRI-O、およびホストからコンテナーまたは双方向のマウント伝達が設定されている他のコンテナー (コンテナー 2 など) からは引き続き表示されます。
- コンテナー 2 とコンテナー 3 の動作は変更されていません。

7.4.2. マウント namespace のカプセル化の設定

クラスターがより少ないリソースオーバーヘッドで実行されるように、マウント namespace のカプセル化を設定できます。

注記

マウント namespace のカプセル化はテクノロジープレビュー機能であり、デフォルトでは無効になっています。これを使用するには、機能を手動で有効にする必要があります。

前提条件

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

手順

1. 次の YAML を使用して、**mount_namespace_config.yaml** という名前のファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-kubens-master
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: kubens.service
---
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-kubens-worker
spec:
  config:
```

```
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: kubens.service
```

2. 次のコマンドを実行して、マウント namespace **MachineConfig** CR を適用します。

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

出力例

```
machineconfig.machineconfiguration.openshift.io/99-kubens-master created
machineconfig.machineconfiguration.openshift.io/99-kubens-worker created
```

3. **MachineConfig** CR がクラスターに適用されるまで、最大 30 分かかる場合があります。次のコマンドを実行して、**MachineConfig** CR のステータスをチェックできます。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-03d4bc4befb0f4ed3566a2c8f7636751   False     True       False
3              0                   0                   0                     45m
worker   rendered-worker-10577f6ab0117ed1825f8af2ac687ddf   False     True       False
3              1                   1
```

4. 次のコマンドを実行した後、**MachineConfig** CR がすべてのコントロールプレーンとワーカーノードに正常に適用されるまで待ちます。

```
$ oc wait --for=condition=Updated mcp --all --timeout=30m
```

出力例

```
machineconfigpool.machineconfiguration.openshift.io/master condition met
machineconfigpool.machineconfiguration.openshift.io/worker condition met
```

検証

クラスターホストのカプセル化を確認するには、次のコマンドを実行します。

1. クラスターホストへのデバッグシェルを開きます。

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

2. **chroot** セッションを開きます。

```
sh-4.4# chroot /host
```

3. systemd マウント namespace を確認します。

```
sh-4.4# readlink /proc/1/ns/mnt
```

出力例

```
mnt:[4026531953]
```

4. kubelet マウント namespace をチェックします。

```
sh-4.4# readlink /proc/$(pgrep kubelet)/ns/mnt
```

出力例

```
mnt:[4026531840]
```

5. CRI-O マウント namespace を確認します。

```
sh-4.4# readlink /proc/$(pgrep crio)/ns/mnt
```

出力例

```
mnt:[4026531840]
```

これらのコマンドは、systemd、kubelet、およびコンテナーランタイムに関連付けられたマウント namespace を返します。OpenShift Container Platform では、コンテナーランタイムは CRI-O です。

上記の例のように、systemd が kubelet および CRI-O とは異なるマウント namespace にある場合、カプセル化が有効になります。3 つのプロセスすべてが同じマウント namespace にある場合、カプセル化は有効ではありません。

7.4.3. カプセル化された namespace の検査

Red Hat Enterprise Linux CoreOS (RHCOS) で利用可能な **kubensenter** スクリプトを使用して、デバッグまたは監査の目的でクラスターホストオペレーティングシステムの Kubernetes 固有のマウントポイントを検査できます。

クラスターホストへの SSH シェルセッションは、デフォルトの namespace にあります。SSH シェルプロンプトで Kubernetes 固有のマウントポイントを検査するには、ルートとして **kubensenter** スクリプトを実行する必要があります。**kubensenter** スクリプトは、マウントカプセル化の状態を認識しており、カプセル化が有効になっていない場合でも安全に実行できます。

注記

oc debug リモートシェルセッションは、デフォルトで Kubernetes namespace 内で開始されます。**oc debug** を使用する場合、マウントポイントを検査するために **kubensenter** を実行する必要はありません。

カプセル化機能が有効になっていない場合、**kubensenter findmnt** コマンドと **findmnt** コマンドは、**oc debug** セッションで実行されているか SSH シェルプロンプトで実行されているかに関係なく、同じ出力を返します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- クラスターホストへの SSH アクセスを設定しました。

手順

1. クラスターホストへのリモート SSH シェルを開きます。以下に例を示します。

```
$ ssh core@<node_name>
```

2. root ユーザーとして、提供された **kubesenter** スクリプトを使用してコマンドを実行します。Kubernetes namespace 内で単一のコマンドを実行するには、コマンドと任意の引数を **kubenenter** スクリプトに提供します。たとえば、Kubernetes namespace 内で **findmnt** コマンドを実行するには、次のコマンドを実行します。

```
[core@control-plane-1 ~]$ sudo kubensenter findmnt
```

出力例

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
TARGET                                SOURCE                FSTYPE     OPTIONS
/
/dev/sda4[/ostree/deploy/rhcos/deploy/32074f0e8e5ec453e56f5a8a7bc9347eaa4172349ceab9
c22b709d9d71a3f4b0.0]
|                                                   xfs
rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota
                                  shm                 tmpfs
...
```

3. Kubernetes namespace 内で新しいインタラクティブシェルを開始するには、引数を指定せずに **kubesenter** スクリプトを実行します。

```
[core@control-plane-1 ~]$ sudo kubensenter
```

出力例

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
```

7.4.4. カプセル化された namespace で追加サービスを実行する

ホスト OS で実行する機能に依存し、kubelet、CRI-O、またはコンテナー自体によって作成されたマウントポイントを表示できる監視ツールは、これらのマウントポイントを表示するためにコンテナーマウント namespace に入る必要があります。OpenShift Container Platform に付属する **kubensenter** スクリプトは、Kubernetes マウントポイント内で別のコマンドを実行し、既存のツールを適応させるために使用できます。

kubensenter スクリプトは、マウントカプセル化機能の状態を認識しており、カプセル化が有効になっていない場合でも安全に実行できます。その場合、スクリプトはデフォルトのマウント namespace で提供されたコマンドを実行します。

たとえば、systemd サービスを新しい Kubernetes マウント namespace 内で実行する必要がある場合は、サービスファイルを編集し、**kubensenter** で **ExecStart=** コマンドラインを使用します。

```
[Unit]
Description=Example service
[Service]
ExecStart=/usr/bin/kubensenter /path/to/original/command arg1 arg2
```

7.4.5. 関連情報

- namespace とは
- nsenter を使用して namespace 内のコンテナーを管理する
- MachineConfig

第8章 ベアメタルホストの管理

OpenShift Container Platform をベアメタルクラスターにインストールする場合、クラスターに存在するベアメタルホストの **machine** および **machineset** カスタムリソース (CR) を使用して、ベアメタルノードをプロビジョニングし、管理できます。

8.1. ベアメタルホストおよびノードについて

Red Hat Enterprise Linux CoreOS (RHCOS) ベアメタルホストをクラスター内のノードとしてプロビジョニングするには、まずベアメタルホストハードウェアに対応する **MachineSet** カスタムリソース (CR) オブジェクトを作成します。ベアメタルホストコンピュートマシンセットは、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 ラベルをこれらのコンピュートマシンセットに適用してから、インフラストラクチャーコンポーネントを更新して、それらのマシンでのみ実行されるようにします。

Machine CR は、**metal3.io/autoscale-to-hosts** アノテーションを含む関連する **MachineSet** をスケールアップする際に自動的に作成されます。OpenShift Container Platform は **Machine** CR を使用して、**MachineSet** CR で指定されるホストに対応するベアメタルノードをプロビジョニングします。

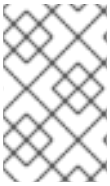
8.2. ベアメタルホストのメンテナンス

OpenShift Container Platform Web コンソールからクラスター内のベアメタルホストの詳細を維持することができます。**Compute → Bare Metal Hosts** に移動し、**Actions** ドロップダウンメニューからタスクを選択します。ここでは、BMC の詳細、ホストの起動 MAC アドレス、電源管理の有効化などの項目を管理できます。また、ホストのネットワークインターフェイスおよびドライブの詳細を確認することもできます。

ベアメタルホストをメンテナンスモードに移行できます。ホストをメンテナンスモードに移行すると、スケジューラーはすべての管理ワークロード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. 関連するコンピュート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

8.2.1. Web コンソールを使用したベアメタルホストのクラスターへの追加

Web コンソールのクラスターにベアメタルホストを追加できます。

前提条件

- RHCOS クラスターのベアメタルへのインストール
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. Web コンソールで、**Compute → Bare Metal Hosts** に移動します。
2. **Add Host → New with Dialog** を選択します。
3. 新規ベアメタルホストの一意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コンピュートマシンセットを使用して、ベアメタルノードの数を管理することもできます。

8.2.2. Web コンソールの YAML を使用したベアメタルホストのクラスターへの追加

ベアメタルホストを記述する YAML ファイルを使用して、Web コンソールのクラスターにベアメタルホストを追加できます。

前提条件

- クラスターで使用するために RHCOS コンピュートマシンをベアメタルインフラストラクチャーにインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- ベアメタルホストの **Secret** CR を作成します。

手順

1. Web コンソールで、**Compute → Bare Metal Hosts** に移動します。
2. **Add Host → New from YAML** を選択します。
3. 以下の YAML をコピーして貼り付け、ホストの詳細で関連フィールドを変更します。

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <bare_metal_host_name>
```

```
spec:
  online: true
  bmc:
    address: <bmc_address>
    credentialsName: <secret_credentials_name> (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は、シークレットについて 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コンピュートマシンセットを使用して、ベアメタルノードの数を管理することもできます。

8.2.3. 利用可能なベアメタルホストの数へのマシンの自動スケーリング

利用可能な **BareMetalHost** オブジェクトの数に一致する **Machine** オブジェクトの数を自動的に作成するには、**metal3.io/autoscale-to-hosts** アノテーションを **MachineSet** オブジェクトに追加します。

前提条件

- クラスターで使用する RHCOS ベアメタルコンピュートマシンをインストールし、対応する **BareMetalHost** オブジェクトを作成します。
- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **metal3.io/autoscale-to-hosts** アノテーションを追加して、自動スケーリング用に設定するコンピュートマシンセットにアノテーションを付けます。**<machineset>** をコンピュートマシンセットの名前に置き換えます。

```
$ oc annotate machineset <machineset> -n openshift-machine-api 'metal3.io/autoscale-to-hosts=<any_value>'
```

新しいスケーリングされたマシンが起動するまで待ちます。

注記

BareMetalHost オブジェクトを使用してクラスター内にマシンを作成し、その後ラベルまたはセレクターが **BareMetalHost** で変更される場合、**BareMetalHost** オブジェクトは **Machine** オブジェクトが作成された **MachineSet** に対して引き続きカウントされます。

8.2.4. プロビジョナーノードからのベアメタルホストの削除

特定の状況では、プロビジョナーノードからベアメタルホストを一時的に削除する場合があります。たとえば、OpenShift Container Platform 管理コンソールを使用して、または Machine Config Pool の更新の結果として、ベアメタルホストの再起動がトリガーされたプロビジョニング中に、OpenShift Container Platform は統合された Dell Remote Access Controller (iDrac) にログインし、ジョブキューの削除を発行します。

利用可能な **BareMetalHost** オブジェクトの数と一致する数の **Machine** オブジェクトを管理しないようにするには、**baremetalhost.metal3.io/detached** アノテーションを **MachineSet** オブジェクトに追加します。

注記

このアノテーションは、**Provisioned**、**ExternallyProvisioned**、または **Ready/Available** 状態の **BareMetalHost** オブジェクトに対してのみ効果があります。

前提条件

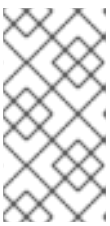
- クラスターで使用する RHCOS ベアメタルコンピュートマシンをインストールし、対応する **BareMetalHost** オブジェクトを作成します。
- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. プロビジョナーノードから削除するコンピューティングマシンセットに、**baremetalhost.metal3.io/detached** アノテーションを追加してアノテーションを付けます。

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached'
```

   新しいマシンが起動するまで待ちます。



   注記

   BareMetalHost オブジェクトを使用してクラスター内にマシンを作成し、その後ラベルまたはセレクターが **BareMetalHost** で変更される場合、**BareMetalHost** オブジェクトは **Machine** オブジェクトが作成された **MachineSet** に対して引き続きカウントされます。

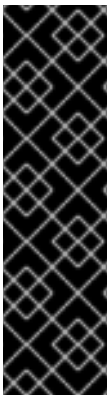
2. プロビジョニングのユースケースでは、次のコマンドを使用して、再起動が完了した後にアノテーションを削除します。

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached-'
```

関連情報

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

第9章 BARE METAL EVENT RELAY を使用したベアメタルイベントのモニタリング



重要

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

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

9.1. ベアメタル イベント

重要

Bare Metal Event Relay Operator は非推奨になりました。Bare Metal Event Relay Operator を使用してベアメタルホストを監視する機能は、今後の OpenShift Container Platform リリースでは削除される予定です。

Bare Metal Event Relay を使用して、OpenShift Container Platform クラスターで実行されるアプリケーションを、基礎となるベアメタルホストで生成されるイベントにサブスクライブします。Redfish サービスは、ノードでイベントをパブリッシュし、サブスクライブされたアプリケーションに高度なメッセージキューでイベントを送信します。

ベアメタルイベントは、Distributed Management Task Force (DMTF) のガイダンスに基づいて開発されたオープン Redfish 標準に基づいています。Redfish は、REST API を使用してセキュアな業界標準プロトコルを提供します。このプロトコルは、分散された、コンバージドまたはソフトウェア定義のリソースおよびインフラストラクチャーの管理に使用されます。

Redfish から公開されるハードウェア関連のイベントには、以下が含まれます。

- 温度制限の違反
- サーバーステータス
- fan ステータス

Bare Metal Event Relay Operator をデプロイし、アプリケーションをサービスにサブスクライブして、ベアメタルイベントの使用を開始します。Bare Metal Event Relay Operator は Redfish ベアメタルイベントサービスのライフサイクルをインストールし、管理します。

注記

Bare Metal Event Relay は、ベアメタルインフラストラクチャーにプロビジョニングされる単一ノードクラスターの Redfish 対応デバイスでのみ機能します。

9.2. ベアメタルイベントの仕組み

Bare Metal Event Relay により、ベアメタルクラスターで実行されるアプリケーションが Redfish ハー

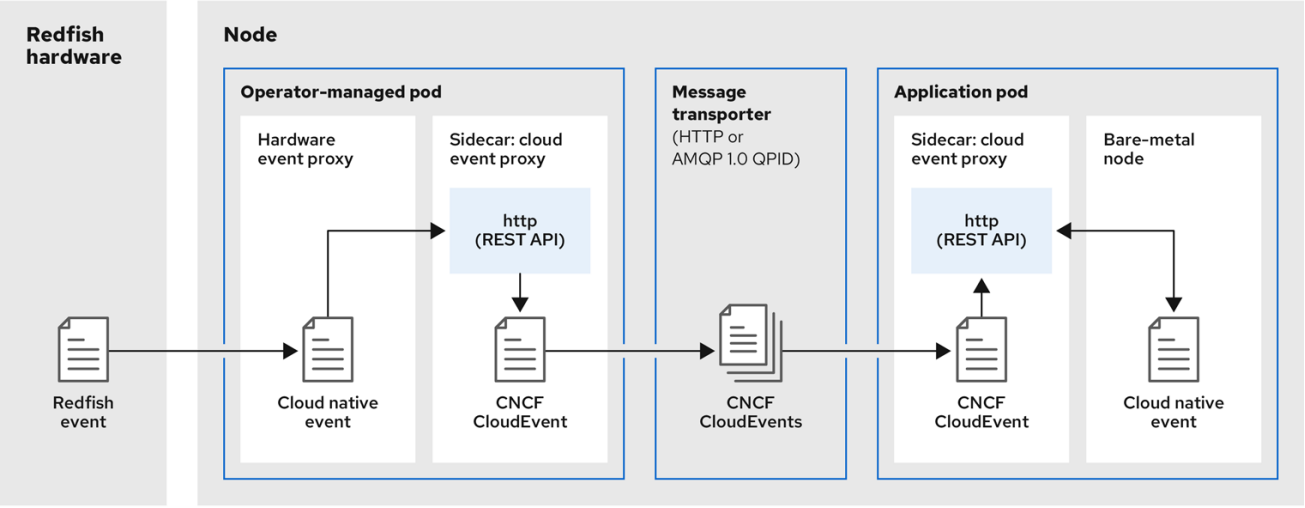
ドウェアの変更や障害に迅速に対応することができます。たとえば、温度のしきい値の違反、fan の障害、ディスク損失、電源停止、メモリー障害などが挙げられます。これらのハードウェアイベントは、HTTP トランスポートまたは AMQP メカニズムを使用して配信されます。メッセージングサービスのレイテンシーは 10 ミリ秒から 20 ミリ秒です。

Bare Metal Event Relay により、ハードウェアイベントでパブリッシュ - サブスクライブサービスを使用できます。アプリケーションは、REST API を使用してイベントをサブスクライブできます。Bare Metal Event Relay は、Redfish OpenAPI v1.8 以降に準拠するハードウェアをサポートします。

9.2.1. Bare Metal Event Relay データフロー

以下の図は、ベアメタルイベントのデータフローの例を示しています。

図9.1 Bare Metal Event Relay データフロー



9.2.1.1. Operator 管理の Pod

Operator はカスタムリソースを使用して、**HardwareEvent** CR を使用して Bare Metal Event Relay およびそのコンポーネントが含まれる Pod を管理します。

9.2.1.2. Bare Metal イベントリレー

起動時に、Bare Metal Event Relay は Redfish API をクエリーし、カスタムレジストリーを含むすべてのメッセージレジストリーをダウンロードします。その後、Bare Metal Event Relay は Redfish ハードウェアからサブスクライブされたイベントを受信し始めます。

Bare Metal Event Relay により、ベアメタルクラスターで実行されるアプリケーションが Redfish ハードウェアの変更や障害に迅速に対応することができます。たとえば、温度のしきい値の違反、fan の障害、ディスク損失、電源停止、メモリー障害などが挙げられます。イベントは **HardwareEvent** CR を使用してレポートされます。

9.2.1.3. クラウドネイティブイベント

クラウドネイティブイベント (CNE) は、イベントデータの形式を定義する REST API 仕様です。

9.2.1.4. CNCF CloudEvents

CloudEvents は、イベントデータの形式を定義するために Cloud Native Computing Foundation (CNCF) によって開発されたベンダーに依存しない仕様です。

9.2.1.5. HTTP トランスポートまたは AMQP ディスパッチルーター

HTTP トランスポートまたは AMQP ディスパッチルーターは、パブリッシャーとサブスクライバー間のメッセージ配信サービスを行います。

注記

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

9.2.1.6. クラウドイベントプロキシーサイドカー

クラウドイベントプロキシーサイドカーコンテナーイメージは O-RAN API 仕様をベースとしており、ハードウェアイベントのパブリッシュ - サブスクライブイベントフレームワークを提供します。

9.2.2. サービスを解析する Redfish メッセージ

Bare Metal Event Relay は Redfish イベントを処理する他に、**Message** プロパティーなしでイベントのメッセージ解析を提供します。プロキシーは、起動時にハードウェアからベンダー固有のレジストリーを含むすべての Redfish メッセージブローカーをダウンロードします。イベントに **Message** プロパティーが含まれていない場合、プロキシーは Redfish メッセージレジストリーを使用して **Message** プロパティーおよび **Resolution** プロパティーを作成し、イベントをクラウドイベントフレームワークに渡す前にイベントに追加します。このサービスにより、Redfish イベントでメッセージサイズが小さくなり、送信レイテンシーが短縮されます。

9.2.3. CLI を使用した Bare Metal Event リレーのインストール

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

前提条件

- RedFish 対応ベースボード管理コントローラー (BMC) を持つノードでベアメタルハードウェアにインストールされるクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. Bare Metal Event Relay の namespace を作成します。
   a. 以下の YAML を **bare-metal-events-namespace.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: openshift-bare-metal-events
  labels:
    name: openshift-bare-metal-events
    openshift.io/cluster-monitoring: "true"
```

b. **namespace** CR を作成します。

```
$ oc create -f bare-metal-events-namespace.yaml
```

2. Bare Metal Event Relay Operator の Operator グループを作成します。

a. 以下の YAML を **bare-metal-events-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: bare-metal-event-relay-group
  namespace: openshift-bare-metal-events
spec:
  targetNamespaces:
  - openshift-bare-metal-events
```

b. **OperatorGroup** CR を作成します。

```
$ oc create -f bare-metal-events-operatorgroup.yaml
```

3. Bare Metal Event Relay にサブスクライブします。

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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bare-metal-event-relay-subscription
  namespace: openshift-bare-metal-events
spec:
  channel: "stable"
  name: bare-metal-event-relay
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. **Subscription** CR を作成します。

```
$ oc create -f bare-metal-events-sub.yaml
```

検証

Bare Metal Event Relay Operator がインストールされていることを確認するには、以下のコマンドを実行します。

```
$ oc get csv -n openshift-bare-metal-events -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

9.2.4. Web コンソールを使用した Bare Metal Event リレーのインストール

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

前提条件

- RedFish 対応ベースボード管理コントローラー (BMC) を持つノードでベアメタルハードウェアにインストールされるクラスター。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Container Platform Web コンソールを使用して Bare Metal Event Relay をインストールします。
   a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
   b. 利用可能な Operator のリストから **Bare Metal Event Relay**を選択し、**Install** をクリックします。
   c. **Install Operator**ページで、**Namespace** を選択または作成し、**openshift-bare-metal-events** を選択して、**Install** をクリックします。

検証

オプション: 以下のチェックを実行して、Operator が正常にインストールされていることを確認できます。

1. **Operators → Installed Operators** ページに切り替えます。
2. **Status** が **InstallSucceeded** の状態で、**Bare Metal Event Relay**がプロジェクトにリスト表示されていることを確認します。

注記

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

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

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

9.3. AMQ メッセージングバスのインストール

ノードのパブリッシャーとサブスクライバー間で Redfish ベアメタルイベント通知を渡すには、ノード上でローカルを実行するように AMQ メッセージングバスをインストールし、設定できます。これは、クラスターで使用するために AMQ Interconnect Operator をインストールして行います。

注記

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

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

- AMQ Interconnect Operator を独自の **amq-interconnect** namespace にインストールします。AMQ Interconnect Operator のインストール について参照してください。

検証

1. AMQ Interconnect Operator が利用可能で、必要な Pod が実行されていることを確認します。

```
$ oc get pods -n amq-interconnect
```

出力例

```
NAME                                READY   STATUS    RESTARTS   AGE
amq-interconnect-645db76c76-k8ghs       1/1     Running   0          23h
interconnect-operator-5cb5fc7cc-4v7qm   1/1     Running   0          23h
```

2. 必要な **bare-metal-event-relay** ベアメタルイベントプロデューサー Pod が **openshift-bare-metal-events** namespace で実行されていることを確認します。

```
$ oc get pods -n openshift-bare-metal-events
```

出力例

```
NAME                                                READY   STATUS    RESTARTS   AGE
hw-event-proxy-operator-controller-manager-74d5649b7c-dzgtl    2/2     Running   0
25s
```

9.4. クラスターノードの REDFISH BMC ベアメタルイベントのサブスクライブ

ノードの **BMCEventSubscription** カスタムリソース (CR) の作成、イベント用の **HardwareEvent** CR の作成、BMC の **Secret** CR の作成を行うことで、クラスター内のノードで生成される Redfish BMC イベントにサブスクライブできます。

9.4.1. ベアメタルイベントのサブスクライブ

ベースボード管理コントローラー (BMC) を設定して、ベアメタルイベントを OpenShift Container Platform クラスターで実行されているサブスクライブされたアプリケーションに送信できます。Redfish ベアメタルイベントの例には、デバイス温度の増加やデバイスの削除が含まれます。REST API を使用して、アプリケーションをベアメタルイベントにサブスクライブします。

重要

BMCEventSubscription カスタムリソース (CR) は、Redfish をサポートし、ベンダーインターフェイスが **redfish** または **idrac-redfish** に設定されている物理ハードウェアにのみ作成できます。

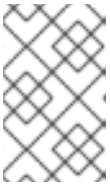
注記

BMCEventSubscription CR を使用して事前定義された Redfish イベントにサブスクライブします。Redfish 標準は、特定のアラートおよびしきい値を作成するオプションを提供しません。例えば、エンクロージャーの温度が摂氏 40 度を超えたときにアラートイベントを受け取るには、ベンダーの推奨に従ってイベントを手動で設定する必要があります。

BMCEventSubscription CR を使用してノードのベアメタルイベントをサブスクライブするには、以下の手順を行います。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- BMC のユーザー名およびパスワードを取得します。
- クラスターに Redfish が有効な Baseboard Management Controller (BMC) を持つベアメタルノードをデプロイし、BMC で Redfish イベントを有効にします。



注記

特定のハードウェアで Redfish イベントを有効にすることは、この情報の対象範囲外です。特定のハードウェアの Redfish イベントを有効にする方法は、BMC の製造元のドキュメントを参照してください。

手順

1. 以下の **curl** コマンドを実行して、ノードのハードウェアで Redfish **EventService** が有効になっていることを確認します。

```
$ curl https://<bmc_ip_address>/redfish/v1/EventService --insecure -H 'Content-Type:
application/json' -u "<bmc_username>:<password>"
```

ここでは、以下のようになります。

bmc_ip_address
: Redfish イベントが生成される BMC の IP アドレスです。

出力例

```
{
  "@odata.context": "/redfish/v1/$metadata#EventService.EventService",
  "@odata.id": "/redfish/v1/EventService",
  "@odata.type": "#EventService.v1_0_2.EventService",
  "Actions": {
    "#EventService.SubmitTestEvent": {
      "EventType@Redfish.AllowableValues": ["StatusChange", "ResourceUpdated",
"ResourceAdded", "ResourceRemoved", "Alert"],
      "target": "/redfish/v1/EventService/Actions/EventService.SubmitTestEvent"
    }
  },
  "DeliveryRetryAttempts": 3,
  "DeliveryRetryIntervalSeconds": 30,
  "Description": "Event Service represents the properties for the service",
  "EventTypesForSubscription": ["StatusChange", "ResourceUpdated", "ResourceAdded",
"ResourceRemoved", "Alert"],
  "EventTypesForSubscription@odata.count": 5,
  "Id": "EventService",
  "Name": "Event Service",
  "ServiceEnabled": true,
  "Status": {
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
  },
  "Subscriptions": {
    "@odata.id": "/redfish/v1/EventService/Subscriptions"
  }
}
```

2. 以下のコマンドを実行して、クラスターの Bare Metal Event Relay サービスのルートを取得します。

```
$ oc get route -n openshift-bare-metal-events
```

出力例

```
NAME           HOST/PORT   PATH                                                    SERVICES
PORT   TERMINATION   WILDCARD
hw-event-proxy             hw-event-proxy-openshift-bare-metal-events.apps.compute-
1.example.com   hw-event-proxy-service   9087   edge          None
```

3. **BMCEventSubscription** リソースを作成し、Redfish イベントにサブスクライブします。

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

```
apiVersion: metal3.io/v1alpha1
kind: BMCEventSubscription
metadata:
  name: sub-01
  namespace: openshift-machine-api
spec:
```

```
hostName: <hostname> 1
destination: <proxy_service_url> 2
context: ''
```

1. Redfish イベントが生成されるワーカーノードの名前または UUID を指定します。
2. ベアメタルイベントプロキシーサービスを指定します (例: **https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook**)。

b. **BMCEventSubscription** CR を作成します。

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

4. オプション: BMC イベントサブスクリプションを削除するには、以下のコマンドを実行します。

```
$ oc delete -f bmc_sub.yaml
```

5. オプション:**BMCEventSubscription** CR を作成せずに Redfish イベントサブスクリプションを手動で作成するには、BMC のユーザー名およびパスワードを指定して以下の **curl** コマンドを実行します。

```
$ curl -i -k -X POST -H "Content-Type: application/json"  -d '{"Destination":
"https://<proxy_service_url>", "Protocol" : "Redfish", "EventTypes": ["Alert"], "Context":
"root"}' -u <bmc_username>:<password>
'https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions' –v
```

ここでは、以下のようになります。

proxy_service_url

ベアメタルイベントプロキシーサービスです (例: **https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook**)。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

出力例

```
HTTP/1.1 201 Created
Server: AMI MegaRAC Redfish Service
Location: /redfish/v1/EventService/Subscriptions/1
Allow: GET, POST
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: X-Auth-Token
Access-Control-Allow-Headers: X-Auth-Token
Access-Control-Allow-Credentials: true
Cache-Control: no-cache, must-revalidate
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>; rel=describedby
Link: <http://redfish.dmtf.org/schemas/v1/EventDestination.v1_6_0.json>
Link: </redfish/v1/EventService/Subscriptions>; path=
ETag: "1651135676"
Content-Type: application/json; charset=UTF-8
```

```
OData-Version: 4.0
Content-Length: 614
Date: Thu, 28 Apr 2022 08:47:57 GMT
```

9.4.2. curl を使用した Redfish ベアメタルイベントサブスクリプションのクエリー

一部のハードウェアベンダーは Redfish ハードウェアイベントサブスクリプションの量を制限します。**curl** を使用して Redfish イベントサブスクリプションの数をクエリーできます。

前提条件

- BMC のユーザー名およびパスワードを取得します。
- クラスターに Redfish が有効な Baseboard Management Controller (BMC) を持つベアメタルノードをデプロイし、BMC で Redfish ハードウェアイベントを有効にします。

手順

1. 以下の **curl** コマンドを実行して、BMC の現在のサブスクリプションを確認します。

```
$ curl --globoff -H "Content-Type: application/json" -k -X GET --user <bmc_username>:
<password> https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions
```

ここでは、以下のようになります。

bmc_ip_address
: Redfish イベントが生成される BMC の IP アドレスです。

出力例

```
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 435 100 435 0 0 399 0 0:00:01 0:00:01 --:--:-- 399
{
  "@odata.context":
"/redfish/v1/$metadata#EventDestinationCollection.EventDestinationCollection",
  "@odata.etag": ""
  1651137375 "",
  "@odata.id": "/redfish/v1/EventService/Subscriptions",
  "@odata.type": "#EventDestinationCollection.EventDestinationCollection",
  "Description": "Collection for Event Subscriptions",
  "Members": [
  {
    "@odata.id": "/redfish/v1/EventService/Subscriptions/1"
  }],
  "Members@odata.count": 1,
  "Name": "Event Subscriptions Collection"
}
```

この例では、サブスクリプションが 1 つ設定されています (**/redfish/v1/EventService/Subscriptions/1**)。

2. オプション: **curl** で **/redfish/v1/EventService/Subscriptions/1** サブスクリプションを削除するには、BMC のユーザー名およびパスワードを指定して以下のコマンドを実行します。

```
$ curl --globoff -L -w "%{http_code} %{url_effective}\n" -k -u <bmc_username>:<password >-
H "Content-Type: application/json" -d '{}' -X DELETE
https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions/1
```

ここでは、以下のようになります。

bmc_ip_address

Redfish イベントが生成される BMC の IP アドレスです。

9.4.3. ベアメタルイベントおよびシークレット CR の作成

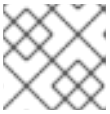
ベアメタルイベントの使用を開始するには、Redfish ハードウェアが存在するホストの **HardwareEvent** カスタムリソース (CR) を作成します。ハードウェアイベントと障害は **hw-event-proxy** ログに報告されます。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- Bare Metal Event Relay をインストールしている。
- BMC Redfish ハードウェア用の **BMCEventSubscription** CR を作成している。

手順

1. **HardwareEvent** カスタムリソース (CR) を作成します。



注記

複数の **HardwareEvent** リソースは許可されません。

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

```
apiVersion: "event.redhat-cne.org/v1alpha1"
kind: "HardwareEvent"
metadata:
  name: "hardware-event"
spec:
  nodeSelector:
    node-role.kubernetes.io/hw-event: "" 1
  logLevel: "debug" 2
  msgParserTimeout: "10" 3
```

1 必須。**nodeSelector** フィールドを使用して、指定されたラベルを持つノードをターゲットにします (例: **node-role.kubernetes.io/hw-event: ""**。

注記

OpenShift Container Platform 4.13 以降でベアメタルイベントに HTTP トランスポートを使用する場合、**HardwareEvent** リソースの **spec.transportHost** フィールドを設定する必要はありません。ベアメタルイベントに AMQP トランスポートを使用する場合にのみ **transportHost** を設定します。

2 オプション: デフォルト値は **debug** です。**hw-event-proxy** ログでログレベルを設定します。**fatal**、**error**、**warning**、**info**、**debug**、**trace** のログレベルを利用できます。

3 オプション: Message Parser のタイムアウト値をミリ秒単位で設定します。メッセージ解析要求がタイムアウト期間内に応答しない場合には、元のハードウェアイベントメッセージはクラウドネイティブイベントフレームワークに渡されます。デフォルト値は 10 です。

b. クラスターで **HardwareEvent** CR を適用します。

```
$ oc create -f hardware-event.yaml
```

2. BMC ユーザー名およびパスワード **Secret** CR を作成します。これにより、ハードウェアイベントプロキシーがベアメタルホストの Redfish メッセージレジストリーにアクセスできるようになります。

a. 以下の YAML を **hw-event-bmc-secret.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: redfish-basic-auth
type: Opaque
stringData: 1
  username: <bmc_username>
  password: <bmc_password>
  # BMC host DNS or IP address
  hostaddr: <bmc_host_ip_address>
```

1 **stringData** の下に、さまざまな項目のプレーンテキスト値を入力します。

b. **Secret** CR を作成します。

```
$ oc create -f hw-event-bmc-secret.yaml
```

関連情報

- ローカルボリュームを使用した永続ストレージ

9.5. ベアメタルイベント REST API リファレンスへのアプリケーションのサブスクライブ

ベアメタルイベント REST API を使用して、親ノードで生成されるベアメタルイベントにアプリケーションをサブスクライブします。

リソースアドレス **/cluster/node/<node_name>/redfish/event** を使用して、アプリケーションを Redfish イベントにサブスクライブします。**<node_name>** は、アプリケーションを実行するクラスターノードに置き換えます。

cloud-event-consumer アプリケーションコンテナーおよび **cloud-event-proxy** サイドカーコンテナーを別のアプリケーション Pod にデプロイします。**cloud-event-consumer** アプリケーションは、アプリケーション Pod の **cloud-event-proxy** コンテナーにサブスクライブします。

次の API エンドポイントを使用して、アプリケーション Pod の **http://localhost:8089/api/ocloudNotifications/v1/** にある **cloud-event-proxy** コンテナーによって投稿された Redfish イベントに **cloud-event-consumer** アプリケーションをサブスクライブします。

- **/api/ocloudNotifications/v1/subscriptions**
  - **POST**: 新しいサブスクリプションを作成します。
  - **GET**: サブスクリプションの一覧を取得します。
- **/api/ocloudNotifications/v1/subscriptions/<subscription_id>**
  - **PUT**: 指定されたサブスクリプション ID に新しいステータス ping 要求を作成します。
- **/api/ocloudNotifications/v1/health**
  - **GET**: **ocloudNotifications** API の正常性ステータスを返します

注記

9089 は、アプリケーション Pod にデプロイされた **cloud-event-consumer** コンテナーのデフォルトポートです。必要に応じて、アプリケーションに異なるポートを設定できます。

api/ocloudNotifications/v1/subscriptions

HTTP メソッド

GET api/ocloudNotifications/v1/subscriptions

説明

サブスクリプションのリストを返します。サブスクリプションが存在する場合は、サブスクリプションの一覧とともに **200 OK** のステータスコードが返されます。

API 応答の例

```
[
 {
  "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri": "http://localhost:9089/api/ocloudNotifications/v1/dummy",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-
8d3a-666c24e34d32",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
 }
]
```

HTTP メソッド

POST api/ocloudNotifications/v1/subscriptions

説明
新しいサブスクリプションを作成します。サブスクリプションが正常に作成されるか、すでに存在する場合は、**201 Created** ステータスコードが返されます。

表9.1 クエリーパラメーター

パラメーター	型
subscription	data

ペイロードの例

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP メソッド
GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

説明
ID が **<subscription_id>**のサブスクリプションの詳細を返します。

表9.2 クエリーパラメーター

パラメーター	型
<subscription_id>	string

API 応答の例

```
{
  "id":"ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "endpointUri":"http://localhost:9089/api/ocloudNotifications/v1/dummy",
  "uriLocation":"http://localhost:8089/api/ocloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
  "resource":"/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/ocloudNotifications/v1/health/

HTTP メソッド
GET api/ocloudNotifications/v1/health/

説明
ocloudNotifications REST API の正常性ステータスを返します。

API 応答の例

```
OK
```

-

9.6. PTP またはベアメタルイベントに HTTP トランスポートを使用するためのコンシューマーアプリケーションの移行

以前に PTP またはベアメタルイベントのコンシューマーアプリケーションをデプロイしている場合は、HTTP メッセージトランスポートを使用するようにアプリケーションを更新する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP Operator または Bare Metal Event Relay を、デフォルトで HTTP トランスポートを使用するバージョン 4.13 以降に更新している。

手順

1. HTTP トランスポートを使用するようにイベントコンシューマーアプリケーションを更新します。クラウドイベントサイドカーデプロイメントの **http-event-publishers** 変数を設定します。
   たとえば、PTP イベントが設定されているクラスターでは、以下の YAML スニペットはクラウドイベントサイドカーデプロイメントを示しています。

```
containers:
  - name: cloud-event-sidecar
    image: cloud-event-sidecar
    args:
      - "--metrics-addr=127.0.0.1:9091"
      - "--store-path=/store"
      - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
      - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043" (1)
      - "--api-port=8089"
```

   (1) PTP Operator は、PTP イベントを生成するホストに対して **NODE_NAME** を自動的に解決します。**compute-1.example.com** はその例です。

   ベアメタルイベントが設定されているクラスターでは、クラウドイベントサイドカーデプロイメント CR で **http-event-publishers** フィールドを **hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043** に設定します。

2. **consumer-events-subscription-service** サービスをイベントコンシューマーアプリケーションと併せてデプロイします。以下に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
```

```
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

第10章 HUGE PAGE の機能およびそれらがアプリケーションによって消費される仕組み

10.1. HUGE PAGE の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 256,000 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トラン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。

10.2. HUGE PAGE がアプリケーションによって消費される仕組み

ノードは、Huge Page の容量をレポートできるように Huge Page を事前に割り当てる必要があります。ノードは、単一サイズの Huge Page のみを事前に割り当てることができます。

Huge Page は、リソース名の **hugepages-<size>** を使用してコンテナ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。

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

Downward API を使用して、コンテナーで使用する 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

10.4. 起動時の HUGE PAGE 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。

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。

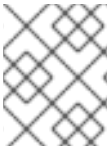
```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-hp
  labels:
    worker-hp: ""
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-hp]}
  nodeSelector:
    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。

10.5. TRANSPARENT HUGE PAGES (THP) の無効化

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

手順

1. 以下の内容でファイルを作成し、**thp-disable-tuned.yaml** という名前を付けます。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: thp-workers-profile
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom tuned profile for OpenShift to turn off THP on worker nodes
      include=openshift-node

      [vm]
      transparent_hugepages=never
    name: openshift-thp-never-worker

  recommend:
  - match:
    - label: node-role.kubernetes.io/worker
    priority: 25
    profile: openshift-thp-never-worker
```

2. Tuned オブジェクトを作成します。

```
$ oc create -f thp-disable-tuned.yaml
```

3. アクティブなプロファイルのリストを確認します。

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

検証

- ノードのいずれかにログインし、通常の THP チェックを実行して、ノードがプロファイルを正常に適用したかどうかを確認します。

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

出力例

```
always madvise [never]
```

第11章 低遅延チューニング

11.1. 低レイテンシーについて

Telco / 5G の領域でのエッジコンピューティングの台頭は、レイテンシーと輻輳を軽減し、アプリケーションのパフォーマンスを向上させる上で重要な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 50 ms の 5G では、レイテンシーの数値を 1 ms 以下にするようにターゲットが設定されます。このレイテンシーの減少により、ワイヤレスのスループットが 10 倍向上します。

Telco 領域にデプロイされるアプリケーションの多くは、ゼロパケットロスに耐えられる低レイテンシーを必要とします。パケットロスをゼロに調整すると、ネットワークのパフォーマンス低下させる固有の問題を軽減することができます。詳細は、Tuning for Zero Packet Loss in Red Hat OpenStack Platform (RHOSP) を参照してください。

エッジコンピューティングの取り組みは、レイテンシーの削減にも役立ちます。クラウドの端にあり、ユーザーに近いと考えてください。これにより、ユーザーと離れた場所にあるデータセンター間の距離が大幅に削減されるため、アプリケーションの応答時間とパフォーマンスのレイテンシーが短縮されます。

管理者は、すべてのデプロイメントを可能な限り低い管理コストで実行できるように、多数のエッジサイトおよびローカルサービスを一元管理できるようにする必要があります。また、リアルタイムの低レイテンシーおよび高パフォーマンスを実現するために、クラスターの特定のノードをデプロイし、設定するための簡単な方法も必要になります。低レイテンシーノードは、Cloud-native Network Functions (CNF) や Data Plane Development Kit (DPDK) などのアプリケーションに役立ちます。

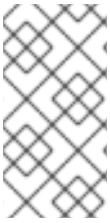
現時点で、OpenShift Container Platform はリアルタイムの実行および低レイテンシーを実現するために OpenShift Container Platform クラスターでソフトウェアを調整するメカニズムを提供します (約 20 マイクロ秒未満の応答時間)。これには、カーネルおよび OpenShift Container Platform の設定値のチューニング、カーネルのインストール、およびマシンの再設定が含まれます。ただし、この方法では 4 つの異なる Operator を設定し、手動で実行する場合に複雑であり、間違いが生じる可能性がある多くの設定を行う必要があります。

OpenShift Container Platform は、ノードチューニング Operator を使用して自動チューニングを実装し、OpenShift Container Platform アプリケーションの低レイテンシーパフォーマンスを実現します。クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更をより容易に実行することができます。管理者は、カーネルを kernel-rt に更新するかどうかを指定し、Pod の infra コンテナ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。

重要

OpenShift Container Platform 4.14 では、クラスターにパフォーマンスプロファイルを適用すると、クラスター内のすべてのノードが再起動します。この再起動には、パフォーマンスプロファイルの対象になっていないコントロールプレーンノードとワーカーノードが含まれます。このリリースでは RHEL 9 と連携した Linux コントロールグループバージョン 2 (cgroup v2) が使用されているため、これは OpenShift Container Platform 4.14 の既知の問題です。パフォーマンスプロファイルに関連付けられた低遅延チューニング機能は cgroup v2 をサポートしていないため、ノードは再起動して cgroup v1 設定に戻ります。

クラスター内のすべてのノードを cgroups v2 設定に戻すには、**Node** リソースを編集する必要があります。(OCPBUGS-16976)



注記

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

OpenShift Container Platform は、さまざまな業界環境の要求を満たすように **PerformanceProfile** を調整できる Node Tuning Operator のワークロードヒントもサポートします。ワークロードのヒントは、**highPowerConsumption** (消費電力が増加する代わりにレイテンシーを非常に低く抑える) と **realTime** (最適なレイテンシーを優先) で利用できます。これらのヒントの **true/false** 設定の組み合わせを使用して、アプリケーション固有のワークロードプロファイルと要件を処理できます。

ワークロードのヒントは、業界セクターの設定に対するパフォーマンスの微調整を簡素化します。1 つのサイズですべてに対応するアプローチの代わりに、ワークロードのヒントは、以下を優先するなどの使用パターンに対応できます。

- 低レイテンシー
- リアルタイム機能
- 電力の効率的な使用

理想的な世界では、これらすべてが優先されます。実際の生活では、他の人を犠牲にしてやってくる人もいます。Node Tuning Operator は、ワークロードの期待を認識し、ワークロードの要求をより適切に満たすことができるようになりました。クラスター管理者は、ワークロードがどのユースケースに分類されるかを指定できるようになりました。Node Tuning Operator は、**PerformanceProfile** を使用して、ワークロードのパフォーマンス設定を微調整します。

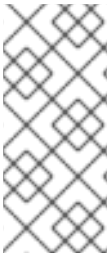
アプリケーションが動作している環境は、その動作に影響を与えます。厳密なレイテンシー要件のない一般的なデータセンターの場合、一部の高性能ワークロード Pod の CPU パーティショニングを可能にする最小限のデフォルトチューニングのみが必要です。レイテンシーが優先されるデータセンターやワークロードの場合でも、消費電力を最適化するための対策が講じられています。最も複雑なケースは、製造機械やソフトウェア無線などのレイテンシーの影響を受けやすい機器に近いクラスターです。この最後のクラスのデプロイメントは、多くの場合、ファーエッジと呼ばれます。ファーエッジデプロイメントの場合、超低レイテンシーが最優先事項であり、電力管理を犠牲にして実現されます。

11.1.1. 低レイテンシーおよびリアルタイムのアプリケーションのハイパースレッディングについて

ハイパースレッディングは、物理 CPU プロセッサーコアが 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

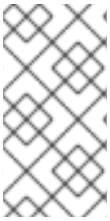
11.2. リアルタイムおよび低レイテンシーワークロードのプロビジョニング

多くの企業や組織は、非常に高性能なコンピューティングを必要としており、とくに金融業界や通信業界では、低い、予測可能なレイテンシーが必要になる場合があります。こうした業界特有の要件に対して、OpenShift Container Platform では、OpenShift Container Platform アプリケーションの低遅延性能と一貫した応答速度を実現するための自動チューニングを実施する Node Tuning Operator を提供しています。

クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更を加えることができます。管理者は、カーネルを kernel-rt (リアルタイム) に更新するか、Pod infra コンテナーを含むクラスターと OS のハウスキーピング業務用に CPU を確保するか、アプリケーションコンテナー用に CPU を分離してワークロードを実行するか、未使用 CPU を無効にして電力消費を抑えるかを指定することができます。

警告

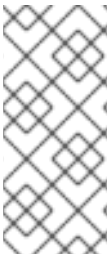
保証された CPU を必要とするアプリケーションと組み合わせて実行プローブを使用すると、レイテンシースパイクが発生する可能性があります。代わりに、適切に設定されたネットワークプローブのセットなど、他のプローブを使用することを推奨します。



注記

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

11.2.1. リアルタイムの既知の制限

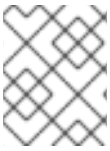


注記

ほとんどのデプロイメントで、3 つのコントロールプレーンノードと 3 つのワーカーノードを持つ標準クラスターを使用する場合、kernel-rt はワーカーノードでのみサポートされます。OpenShift Container Platform デプロイメントのコンパクトノードと単一ノードには例外があります。単一ノードへのインストールの場合、kernel-rt は単一のコントロールプレーンノードでサポートされます。

リアルタイムモードを完全に使用するには、コンテナ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。

11.2.2. リアルタイム機能のあるワーカーのプロビジョニング

1. オプション: ノードを OpenShift Container Platform クラスターに追加します。システムチューニング用の BIOS パラメーターの設定 を参照してください。

2. **oc**コマンドを使用して、リアルタイム機能を必要とするワーカーノードにラベル**worker-rt**を追加します。

3. リアルタイムノード用の新しいマシン設定プールを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-rt
  labels:
    machineconfiguration.openshift.io/role: worker-rt
spec:
  machineConfigSelector:
    matchExpressions:
      - {
          key: machineconfiguration.openshift.io/role,
          operator: In,
          values: [worker, worker-rt],
        }
  paused: false
```

```
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-rt: ""
```

マシン設定プール worker-rt は、**worker-rt** というラベルを持つノードのグループに対して作成されることに注意してください。

4. ノードロールラベルを使用して、ノードを適切なマシン設定プールに追加します。

注記

リアルタイムワークロードで設定するノードを決定する必要があります。クラスター内のすべてのノード、またはノードのサブセットを設定できます。すべてのノードが専用のマシン設定プールの一部であることを期待する Node Tuning Operator。すべてのノードを使用する場合は、Node Tuning Operator がワーカーノードのロールラベルを指すようにする必要があります。サブセットを使用する場合、ノードを新規のマシン設定プールにグループ化する必要があります。

5. 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。

6. **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
```

7. 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
```

8. 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。

9. すべてが予想通りに機能していることを確認します。

11.2.3. リアルタイムカーネルのインストールの確認

以下のコマンドを使用して、リアルタイムカーネルがインストールされていることを確認します。

```
$ oc get node -o wide
```

4.18.0-305.30.1.rt7.102.el8_4.x86_64 cri-o://1.28.5-99.rhaos4.10.gitc3131de.el8 の文字列を含むロール **worker-rt** を持つワーカーに注意してください。

```
NAME                        STATUS   ROLES          AGE  VERSION                INTERNAL-IP
EXTERNAL-IP   OS-IMAGE                                KERNEL-VERSION
CONTAINER-RUNTIME
rt-worker-0.example.com     Ready    worker,worker-rt   5d17h   v1.28.5
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://1.28.5-99.rhaos4.10.gitc3131de.el8
[...]
```

11.2.4. リアルタイムで機能するワークロードの作成

リアルタイム機能を使用するワークロードを準備するには、以下の手順を使用します。

手順

1. QoS クラスの **Guaranteed** を指定して Pod を作成します。
2. オプション: DPDK の CPU 負荷分散を無効にします。
3. 適切なノードセ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。

11.2.5. QoS クラスの Guaranteed を指定した Pod の作成

QoS クラスの **Guaranteed** が指定されている Pod を作成する際には、以下を考慮してください。

- Pod のすべてのコンテナ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:
  securityContext:
    runAsNonRoot: true
```

```
  seccompProfile:
    type: RuntimeDefault
containers:
- name: qos-demo-ctr
  image: <image-pull-spec>
  resources:
    limits:
      memory: "200Mi"
      cpu: "1"
    requests:
      memory: "200Mi"
      cpu: "1"
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

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。

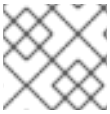
11.2.6. オプション: DPDK 用の CPU 負荷分散の無効化

CPU 負荷分散を無効または有効にする機能は CRI-O レベルで実装されます。CRI-O のコードは、以下の要件を満たす場合にのみ CPU の負荷分散を無効または有効にします。

- Pod は **performance-<profile-name>** ランタイムクラスを使用する必要があります。以下に示すように、パフォーマンスプロファイルのステータスを確認して、適切な名前を取得できます。

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

```
status:
  ...
  runtimeClass: performance-manual
```



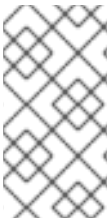
注記

現在、cgroup v2 では CPU 負荷分散の無効化はサポートされていません。

Node Tuning Operator は、関連ノード下での高性能ランタイムハンドラー config snippet の作成と、クラスター下での高性能ランタイムクラスの作成を担当します。これには、 CPU 負荷分散の設定機能を有効にすることを除くと、デフォルトのランタイムハンドラーと同じ内容が含まれます。

Pod の CPU 負荷分散を無効にするには、 **Pod** 仕様に以下のフィールドが含まれる必要があります。

```
apiVersion: v1
kind: Pod
metadata:
  ...
  annotations:
    ...
    cpu-load-balancing.crio.io: "disable"
    ...
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```



注記

CPU マネージャーの静的ポリシーが有効にされている場合に、CPU 全体を使用する Guaranteed QoS を持つ Pod について CPU 負荷分散を無効にします。これ以外の場合に CPU 負荷分散を無効にすると、クラスター内の他のコンテナーのパフォーマンスに影響する可能性があります。

11.2.7. 適切なノードセ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。

11.2.8. リアルタイム機能を備えたワーカーへのワークロードのスケジューリング

Node Tuning Operator によって低レイテンシー用に設定されたマシン設定プールに接続されているノードに一致するラベルセ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。

11.2.9. CPU をオフラインにすることで消費電力を削減

一般に、通信のワークロードを予測できます。すべての CPU リソースが必要なわけではない場合、Node Tuning Operator を使用すると、未使用の CPU をオフラインにして、パフォーマンスプロファイルを手動で更新することにより、消費電力を削減できます。

未使用の CPU をオフラインにするには、次のタスクを実行する必要があります。

1. パフォーマンスプロファイルでオフライン CPU を設定し、YAML ファイルの内容を保存します。

   オフライン CPU を使用したパフォーマンスプロファイルの例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  additionalKernelArgs:
  - nmi_watchdog=0
  - audit=0
  - mce=off
  - processor.max_cstate=1
  - intel_idle.max_cstate=0
  - idle=poll
  cpu:
    isolated: "2-23,26-47"
    reserved: "0,1,24,25"
    offlined: "48-59" (1)
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: single-numa-node
  realTimeKernel:
    enabled: true
```

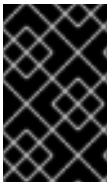
   (1) オプション: **offlined** フィールドに CPU をリストして、指定した CPU をオフラインにすることができます。

2. 次のコマンドを実行して、更新されたプロファイルを適用します。

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

11.2.10. オプション: 省電力設定

優先度の高いワークロードのレイテンシーやスループットに影響を与えることなく、優先度の高いワークロードと同じ場所にある優先度の低いワークロードを持つノードの省電力を有効にすることができます。ワークロード自体を変更することなく、省電力が可能です。



重要

この機能は、Intel Ice Lake 以降の世代の Intel CPU でサポートされています。プロセッサーの機能は、優先度の高いワークロードのレイテンシーとスループットに影響を与える可能性があります。

省電力設定でノードを設定するときは、優先度の高いワークロードを Pod レベルのパフォーマンス設定で設定する必要があります。つまり、Pod で使用されるすべてのコアにその設定が適用されます。

Pod レベルで P ステートと C ステートを無効にすることで、優先度の高いワークロードを設定して、最高のパフォーマンスと最小の待機時間を実現できます。

表11.1 優先度の高いワークロードの設定

アノテーション	設定可能な値	説明
cpu-c-states.crio.io:	• **"enable"** • **"disable"** • **"max_latency:microseconds"**	このアノテーションを使用すると、各 CPU の C ステートを有効または無効にすることができます。あるいは、C ステートの最大レイテンシーをマイクロ秒単位で指定することもできます。たとえば、**cpu-c-states.crio.io**: **"max_latency:10"** を設定して、最大レイテンシー 10 マイクロ秒の C ステートを有効にします。Pod に最高のパフォーマンスを提供するには、値を **"disable"** に設定します。
cpu-freq-governor.crio.io:	サポートされている **cpufreq governor**。	各 CPU の **cpufreq** ガバナーを設定します。**"performance"** ガバナーは、優先度の高いワークロードに推奨されます。

前提条件

- BIOS で C ステートと OS 制御の P ステートを有効にした

手順

1. **per-pod-power-management** を **true** に設定して **PerformanceProfile** を生成します。

```
$ podman run --entrypoint performance-profile-creator -v \
/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 \
--mcp-name=worker-cnf --reserved-cpu-count=20 --rt-kernel=true \
--split-reserved-cpus-across-numa=false --topology-manager-policy=single-numa-node \
--must-gather-dir-path /must-gather -power-consumption-mode=low-latency \ 1
--per-pod-power-management=true > my-performance-profile.yaml
```

1 **per-pod-power-management** が **true** に設定されている場合、**power-consumption-mode** は **default** または **low-latency** である必要があります。

perPodPowerManagement を使用した PerformanceProfile の例

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

```
metadata:
    name: performance
spec:
  [.....]
  workloadHints:
    realTime: true
    highPowerConsumption: false
    perPodPowerManagement: true
```

2. デフォルトの **cpufreq** ガバナーを、**PerformanceProfile** カスタムリソース (CR) で追加のカーネル引数として設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
    name: performance
spec:
  ...
  additionalKernelArgs:
  - cpufreq.default_governor=schedutil (1)
```

(1) **schedutil** ガバナーの使用が推奨されますが、**ondemand** ガバナーや **powersave** ガバナーなどの他のガバナーを使用することもできます。

3. **TunedPerformancePatch** CR で最大 CPU 周波数を設定します。

```
spec:
 profile:
 - data: |
    [sysfs]
    /sys/devices/system/cpu/intel_pstate/max_perf_pct = <x> (1)
```

(1) **max_perf_pct** は、**cpufreq** ドライバーが設定できる最大周波数を、サポートされている最大 CPU 周波数のパーセンテージとして制御します。この値はすべての CPU に適用されます。サポートされている最大周波数は **/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq** で確認できます。開始点として、**All Cores Turbo** 周波数ですべての CPU を制限する割合を使用できます。**All Cores Turbo** 周波数は、すべてのコアがすべて使用されているときに全コアが実行される周波数です。

4. 必要なアノテーションを優先度の高いワークロード Pod に追加します。注釈は **default** 設定を上書きします。

優先度の高いワークロードアノテーションの例

```
apiVersion: v1
kind: Pod
metadata:
 ...
 annotations:
  ...
  cpu-c-states.crio.io: "disable"
  cpu-freq-governor.crio.io: "performance"
```

```
    ...
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```

5. Pod を再起動します。

関連情報

- 推奨されるファームウェア設定の詳細は、vDU クラスターホストの推奨されるファームウェア設定 を参照してください。

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

Node Tuning Operator は、ホスト CPU を、Pod Infra コンテナーを含むクラスターとオペレーティングシステムのハウスキーピング業務用の予約 CPU と、ワークロードを実行するアプリケーションコンテナー用の分離 CPU に分割して管理することができます。これにより、低レイテンシーのワークロード用の CPU を isolated (分離された CPU) として設定できます。

デバイスの割り込みについては、Guaranteed Pod が実行されている CPU を除き、CPU のオーバーロードを防ぐためにすべての分離された CPU および予約された CPU 間で負荷が分散されます。Guaranteed Pod の CPU は、関連するアノテーションが Pod に設定されている場合にデバイス割り込みを処理できなくなります。

パフォーマンスプロファイルで、 **globallyDisableIrqLoadBalancing** は、デバイス割り込みが処理されるかどうかを管理するために使用されます。特定のワークロードでは、予約された CPU は、デバイスの割り込みを処理するのに常に十分な訳ではないため、デバイスの割り込みは分離された CPU でグローバルに無効化されていません。デフォルトでは、Node Tuning Operator は分離された CPU でのデバイス割り込みを無効にしません。

ワークロードの低レイテンシーを確保するには、一部の (すべてではない) Pod で、それらが実行されている CPU がデバイス割り込みを処理しないようにする必要があります。Pod アノテーション **irq-load-balancing.crio.io** は、デバイス割り込みが処理されるかどうかを定義するために使用されます。CRI-O は (設定されている場合)、Pod が実行されている場合にのみデバイス割り込みを無効にします。

11.2.11.1. CPU CFS クォータの無効化

保証された個々の Pod の CPU スロットル調整を減らすには、アノテーション **cpu-quota.crio.io: "disable"** を付けて、Pod 仕様を作成します。このアノテーションは、Pod の実行時に CPU Completely Fair Scheduler (CFS) のクォータを無効にします。次の Pod 仕様には、このアノテーションが含まれています。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
      cpu-quota.crio.io: "disable"
spec:
    runtimeClassName: performance-<profile_name>
...
```

注記

CPU マネージャーの静的ポリシーが有効になっている場合、および CPU 全体を使用する Guaranteed QoS を持つ Pod の場合にのみ、CPU CFS クォータを無効にします。これ以外の場合に CPU CFS クォータを無効にすると、クラスター内の他のコンテナーのパフォーマンスに影響を与える可能性があります。

11.2.11.2. Node Tuning Operator でのグローバルデバイス割り込み処理の無効化

分離された CPU セットのグローバルデバイス割り込みを無効にするように Node Tuning Operator を設定するには、パフォーマンスプロファイルの **globallyDisableIrqLoadBalancing** フィールドを **true** に設定します。**true** の場合、競合する Pod アノテーションは無視されます。**false** の場合、すべての CPU 間で IRQ 負荷が分散されます。

パフォーマンスプロファイルのスニペットは、この設定を示しています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  globallyDisableIrqLoadBalancing: true
...
```

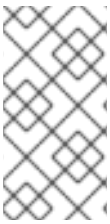
11.2.11.3. 個別の Pod の割り込み処理の無効化

個別の Pod の割り込み処理を無効にするには、パフォーマンスプロファイルで **globalDisableIrqLoadBalancing** が **false** に設定されていることを確認します。次に、Pod 仕様で、**irq-load-balancing.crio.io** Pod アノテーションを **disable** に設定します。次の Pod 仕様には、このアノテーションが含まれています。

```
apiVersion: performance.openshift.io/v2
kind: Pod
metadata:
  annotations:
      irq-load-balancing.crio.io: "disable"
spec:
    runtimeClassName: performance-<profile_name>
...
```

11.2.12. デバイス割り込み処理を使用するためのパフォーマンスプロファイルのアップグレード

Node Tuning Operator パフォーマンスプロファイルのカスタムリソース定義 (CRD) を v1 または v1alpha1 から v2 にアップグレードする場合、**globallyDisableIrqLoadBalancing** は **true** に設定されます。



注記

globallyDisableIrqLoadBalancing は、IRQ ロードバランシングを分離 CPU セットに対して無効にするかどうかを切り替えます。このオプションを **true** に設定すると、分離 CPU セットの IRQ ロードバランシングが無効になります。オプションを **false** に設定すると、IRQ をすべての CPU 間でバランスさせることができます。

11.2.12.1. サポート対象の API バージョン

Node Tuning Operator は、パフォーマンスプロファイル **apiVersion** フィールドの **v2**、**v1**、および **v1alpha1** をサポートします。v1 および v1alpha1 API は同一です。v2 API には、デフォルト値の **false** が設定されたオプションのブール値フィールド **globallyDisableIrqLoadBalancing** が含まれます。

11.2.12.1.1. Node Tuning Operator API の v1alpha1 から v1 へのアップグレード

Node Tuning Operator API バージョンを v1alpha1 から v1 にアップグレードする場合、v1alpha1 パフォーマンスプロファイルは None 変換ストラテジーを使用してオンザフライで変換され、API バージョン v1 の Node Tuning Operator に提供されます。

11.2.12.1.2. Node Tuning Operator API の v1alpha1 または v1 から v2 へのアップグレード

古い Node Tuning Operator API バージョンからアップグレードする場合、既存の v1 および v1alpha1 パフォーマンスプロファイルは、**globallyDisableIrqLoadBalancing** フィールドに **true** の値を挿入する変換 Webhook を使用して変換されます。

11.3. パフォーマンスプロファイルによる低レイテンシーを実現するためのノードのチューニング

パフォーマンスプロファイルを使用すると、特定のマシン設定プールに属するノードのレイテンシーの調整を制御できます。設定を指定すると、**PerformanceProfile** オブジェクトは実際のノードレベルのチューニングを実行する複数のオブジェクトにコンパイルされます。

- ノードを操作する **MachineConfig** ファイル。
- Topology Manager、CPU マネージャー、および OpenShift Container Platform ノードを設定する **KubeletConfig** ファイル。
- Node Tuning Operator を設定する Tuned プロファイル。

パフォーマンスプロファイルを使用して、カーネルを kernel-rt に更新して Huge Page を割り当て、ハウスキーピングデータの実行やワークロードの実行用に CPU をパーティションに分割するかどうかを指定できます。

重要

OpenShift Container Platform 4.15 では、クラスターにパフォーマンスプロファイルを適用すると、クラスター内のすべてのノードが再起動します。この再起動には、パフォーマンスプロファイルの対象になっていないコントロールプレーンノードとワーカーノードが含まれます。このリリースでは、RHEL 9 と連携した Linux コントロールグループバージョン 2 (cgroup v2) が使用されているため、これは OpenShift Container Platform 4.15 の既知の問題です。パフォーマンスプロファイルに関連付けられた低遅延チューニング機能は cgroup v2 をサポートしていないため、ノードは再起動して cgroup v1 設定に戻ります。

クラスター内のすべてのノードを cgroups v2 設定に戻すには、**Node** リソースを編集する必要があります。(OCPBUGS-16976)

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。

11.3.1. Huge Page の設定

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

OpenShift Container Platform は、Huge Page を作成し、割り当てる方法を提供します。Node Tuning Operator は、パフォーマンスプロファイルを使用して、これをより簡単に行う方法を提供します。

たとえば、パフォーマンスプロファイルの **hugepages pages** セクションで、**size**、**count**、およびオプションで **node** の複数のブロックを指定できます。

```
hugepages:
  defaultHugepagesSize: "1G"
  pages:
  - size:  "1G"
    count:  4
    node:  0 1
```

1 **node** は、Huge Page が割り当てられる NUMA ノードです。 **node** を省略すると、ページはすべての NUMA ノード間で均等に分散されます。

注記

更新が完了したことを示す関連するマシン設定プールのステータスを待機します。

これらは、Huge Page を割り当てるのに必要な唯一の設定手順です。

検証

- 設定を確認するには、ノード上の **/proc/meminfo** ファイルを参照します。

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

```
# grep -i huge /proc/meminfo
```

出力例

```
AnonHugePages:    ###### ##
ShmemHugePages:        0 kB
HugePages_Total:       2
HugePages_Free:        2
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       #### ##
Hugetlb:            #### ##
```

- 新規サイズを報告するには、**oc describe** を使用します。

```
$ oc describe node worker-0.ocp4poc.example.com | grep -i huge
```

出力例

```
                            hugepages-1g=true
 hugepages-###:  ###
 hugepages-###:  ###
```

11.3.2. 複数の Huge Page サイズの割り当て

同じコンテナ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、Node Tuning Operator は以下に示すようにノード上に両方のサイズを設定します。

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

11.3.3. IRQ 動的負荷分散用ノードの設定

どのコアがデバイス割り込み要求 (IRQ) を受信できるかを制御するために、IRQ 動的負荷分散用にクラスターノードを設定します。

前提条件

- コアを分離するには、すべてのサーバーハードウェアコンポーネントが IRQ アフィニティーをサポートしている必要があります。サーバーのハードウェアコンポーネントが IRQ アフィニティーをサポートしているかどうかを確認するには、サーバーのハードウェア仕様を参照するか、ハードウェアプロバイダーに問い合わせてください。

手順

1. cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。
2. パフォーマンスプロファイルの **apiVersion** を **performance.openshift.io/v2** を使用するように設定します。
3. **globallyDisableIrqLoadBalancing** フィールドを削除するか、これを **false** に設定します。
4. 適切な分離された CPU と予約された CPU を設定します。以下のスニペットは、2 つの CPU を確保するプロファイルを示しています。IRQ 負荷分散は、**isolated** CPU セットで実行されている Pod について有効にされます。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: dynamic-irq-profile
spec:
  cpu:
    isolated: 2-5
    reserved: 0-1
...
```

注記

予約および分離された CPU を設定する場合に、Pod 内の infra コンテナーは予約された 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: dynamic-irq-pod
    image: "registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15"
    command: ["sleep", "10h"]
    resources:
      requests:
        cpu: 2
        memory: "200M"
      limits:
        cpu: 2
        memory: "200M"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  runtimeClassName: performance-dynamic-irq-profile
# ...
```

6. performance-<profile_name> の形式で Pod **runtimeClassName** を入力します。ここで、<profile_name> は **PerformanceProfile** YAML の **name** です (例: **performance-dynamic-irq-profile**)。

7. ノードセレクターを cnf-worker をターゲットに設定するように設定します。

8. Pod が正常に実行されていることを確認します。ステータスが **running** であり、正しい cnf-worker ノードが設定されている必要があります。

```
$ oc get pod -o wide
```

予想される出力

```
NAME          READY   STATUS    RESTARTS   AGE    IP          NODE
```

```
NOMINATED NODE   READINESS GATES
dynamic-irq-pod   1/1    Running   0          5h33m   <ip-address>   <node-name>   <none>
<none>
```

9. IRQ の動的負荷分散向けに設定された Pod が実行される CPU を取得します。

```
$ oc exec -it dynamic-irq-pod -- /bin/bash -c "grep Cpus_allowed_list /proc/self/status | awk
'{print $2}'"
```

予想される出力

```
Cpus_allowed_list:  2-3
```

10. ノードの設定が正しく適用されていることを確認します。ノードにログインして設定を確認します。

```
$ oc debug node/<node-name>
```

予想される出力

```
Starting pod/<node-name>-debug ...
To use host binaries, run `chroot /host`

Pod IP: <ip-address>
If you don't see a command prompt, try pressing enter.

sh-4.4#
```

11. ノードのファイルシステムを使用できることを確認します。

```
sh-4.4# chroot /host
```

予想される出力

```
sh-4.4#
```

12. デフォルトのシステム CPU アフィニティーマスクに **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
```

11.3.4. IRQ アフィニティー設定のサポートについて

一部の IRQ コントローラーでは IRQ アフィニティー設定がサポートされていないため、常にすべてのオンライン CPU が IRQ マスクとして公開されます。これらの IRQ コントローラーは CPU 0 で正常に実行されます。

以下は、IRQ アフィニティー設定がサポートされていないことを Red Hat が認識しているドライバーとハードウェアの例です。このリストはすべてを網羅しているわけではありません。

- **megaraid_sas** などの一部の RAID コントローラードライバー
- 多くの不揮発性メモリーエクスプレス (NVMe) ドライバー
- 一部の LAN on Motherboard (LOM) ネットワークコントローラー
- **managed_irqs** を使用するドライバー

注記

IRQ アフィニティー設定をサポートしない理由は、プロセッサーの種類、IRQ コントローラー、マザーボードの回路接続などに関連している可能性があります。

分離された CPU に有効な IRQ アフィニティーが設定されている場合は、一部のハードウェアまたはドライバーで IRQ アフィニティー設定がサポートされていないことを示唆している可能性があります。有効なアフィニティーを見つけるには、ホストにログインし、次のコマンドを実行します。

```
$ find /proc/irq -name effective_affinity -printf "%p: " -exec cat {} \;
```

出力例

出力例

```
/proc/irq/0/effective_affinity: 1
/proc/irq/1/effective_affinity: 8
/proc/irq/2/effective_affinity: 0
/proc/irq/3/effective_affinity: 1
/proc/irq/4/effective_affinity: 2
/proc/irq/5/effective_affinity: 1
/proc/irq/6/effective_affinity: 1
/proc/irq/7/effective_affinity: 1
/proc/irq/8/effective_affinity: 1
/proc/irq/9/effective_affinity: 2
/proc/irq/10/effective_affinity: 1
/proc/irq/11/effective_affinity: 1
/proc/irq/12/effective_affinity: 4
/proc/irq/13/effective_affinity: 1
/proc/irq/14/effective_affinity: 1
/proc/irq/15/effective_affinity: 1
/proc/irq/24/effective_affinity: 2
/proc/irq/25/effective_affinity: 4
/proc/irq/26/effective_affinity: 2
/proc/irq/27/effective_affinity: 1
/proc/irq/28/effective_affinity: 8
/proc/irq/29/effective_affinity: 4
/proc/irq/30/effective_affinity: 4
/proc/irq/31/effective_affinity: 8
/proc/irq/32/effective_affinity: 8
/proc/irq/33/effective_affinity: 1
/proc/irq/34/effective_affinity: 2
```

一部のドライバーは、**managed_irqs** を使用します。そのアフィニティーはカーネルによって内部的に管理され、ユーザー空間はアフィニティーを変更できません。場合によっては、これらの IRQ が分離された CPU に割り当てられることもあります。**manage_irqs** の詳細については、Affinity of managed interrupts cannot be changed even if they target isolated CPU を参照してください。

11.3.5. クラスターのハイパースレッディングの設定

OpenShift Container Platform クラスターのハイパースレッディングを設定するには、パフォーマンスプロファイルの CPU スレッドを、予約または分離された CPU プールに設定された同じコアに設定します。

注記

パフォーマンスプロファイルを設定してから、ホストのハイパースレッディング設定を変更する場合は、新規の設定に一致するように **PerformanceProfile** YAML の CPU の **isolated** および **reserved** フィールドを更新するようにしてください。

警告

以前に有効にされたホストのハイパースレッディング設定を無効にすると、**PerformanceProfile** YAML にリスト表示されている CPU コア ID が正しくなくなる可能性があります。この設定が間違っていると、リスト表示される CPU が見つからなくなるため、ノードが利用できなくなる可能性があります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (oc) のインストール。

手順

1. 設定する必要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。

11.3.5.1. 低レイテンシーアプリケーションのハイパースレッディングの無効化

低レイテンシー処理用にクラスターを設定する場合、クラスターをデプロイする前にハイパースレッディングを無効にするかどうかを考慮してください。ハイパースレッディングを無効にするには、以下を実行します。

1. ハードウェアとトポロジーに適したパフォーマンスプロファイルを作成します。
2. **nosmt** を追加のカーネル引数として設定します。以下のパフォーマンスプロファイルの例は、この設定について示しています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - idle=poll
    - intel_idle.max_cstate=0
    - nosmt
  cpu:
    isolated: 2-3
    reserved: 0-1
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 2
        node: 0
        size: 1G
  nodeSelector:
```

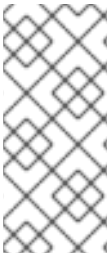
```
    node-role.kubernetes.io/performance: ''
  realTimeKernel:
    enabled: true
```

注記

予約および分離された CPU を設定する場合に、Pod 内の infra コンテナーは予約された 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。

11.3.6. ワークロードのヒントを理解する

次の表は、消費電力とリアルタイム設定の組み合わせがレイテンシーにどのように影響するかを示しています。



注記

次のワークロードヒントは手動で設定できます。Performance Profile Creator を使用して、ワークロードのヒントを操作することもできます。パフォーマンスプロファイルの詳細については、「パフォーマンスプロファイルの作成」セクションを参照してください。ワークロードヒントが手動で設定され、**realTime** ワークロードヒントが明示的に設定されていない場合は、デフォルトで **true** に設定されます。

パフォーマンスプロファイル作成者の設定	Hint	環境	説明
デフォルト	workloadHints: highPowerConsumption: false realTime: false	レイテンシー要件のない高スループットクラスター	CPU パーティショニングのみで達成されるパフォーマンス。
Low-latency	workloadHints: highPowerConsumption: false realTime: true	地域のデータセンター	エネルギー節約と低レイテンシーの両方が望ましい: 電力管理、レイテンシー、スループットの間の妥協。
Ultra-low-latency	workloadHints: highPowerConsumption: true realTime: true	ファーエッジクラスター、レイテンシークリティカルなワークロード	消費電力の増加を犠牲にして、絶対的な最小のレイテンシーと最大の決定論のために最適化されています。

パフォーマンスプロファイル作成者の設定	Hint	環境	説明
Pod ごとの電源管理	workloadHints: realTime: true highPowerConsumption: false perPodPowerManagement: true	重要なワークロードと重要でないワークロード	Pod ごとの電源管理が可能です。

関連情報

- Performance Profile Creator (PPC) を使用してパフォーマンスプロファイルを生成する方法の詳細は、Creating a performance profile を参照してください。

11.3.7. ワークロードヒントを手動で設定する

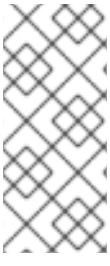
手順

1. ワークロードのヒントについての表の説明に従って、環境のハードウェアとトポロジーに適した **PerformanceProfile** を作成します。予想されるワークロードに一致するようにプロファイルを調整します。この例では、可能な限り低いレイテンシーに調整します。
2. **highPowerConsumption** および **realTime** ワークロードのヒントを追加します。ここでは両方とも **true** に設定されています。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: workload-hints
spec:
  ...
  workloadHints:
    highPowerConsumption: true (1)
    realTime: true (2)
```

(1) **highPowerConsumption** が **true** の場合、ノードは非常に低いレイテンシーに調整されますが、消費電力が増加します。

(2) システムの待ち時間に影響を与える可能性のある一部のデバッグおよび監視機能を無効にします。



注記

パフォーマンスプロファイルで **realTime** ワークロードヒントフラグが **true** に設定されている場合は、固定された CPU を持つすべての保証された Pod に **cpu-quota.crio.io: disable** アノテーションを追加します。このアノテーションは、Pod 内のプロセスのパフォーマンスの低下を防ぐために必要です。**realTime** ワークロードヒントが明示的に設定されていない場合は、デフォルトで **true** に設定されます。

関連情報

- 個々の保証された Pod の CPU スロットルを減らす方法は、CPU CFS クォータの無効化 を参照してください。

11.3.8. infra およびアプリケーションコンテナーの 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、Node Tuning Operator を使用してノードを調整した後、CPU でプロセスがどのように実行されるかを示しています。

表11.2 プロセスの CPU 割り当て

プロセスタイプ	Details
Burstable および **BestEffort** Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
インフラストラクチャー Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
割り込み	予約済み CPU にリダイレクトします (OpenShift Container Platform 4.7 以降ではオプション)
カーネルプロセス	予約済み CPU へのピン
レイテンシーの影響を受けやすいワークロード Pod	分離されたプールからの排他的 CPU の特定のセットへのピン
OS プロセス/systemd サービス	予約済み CPU へのピン

すべての QoS プロセスタイプ (**Burstable**、**BestEffort**、または **Guaranteed**) の Pod に割り当て可能なノード上のコアの容量は、分離されたプールの容量と同じです。予約済みプールの容量は、クラスターおよびオペレーティングシステムのハウスキーピング業務で使用するためにノードの合計コア容量から削除されます。

例 1

ノードは 100 コアの容量を備えています。クラスター管理者は、パフォーマンスプロファイルを使用して、50 コアを分離プールに割り当て、50 コアを予約プールに割り当てます。クラスター管理者は、25 コアを QoS **Guaranteed** Pod に割り当て、25 コアを **BestEffort** または **Burstable** Pod に割り当てます。これは、分離されたプールの容量と一致します。

例 2

ノードは 100 コアの容量を備えています。クラスター管理者は、パフォーマンスプロファイルを使用して、50 コアを分離プールに割り当て、50 コアを予約プールに割り当てます。クラスター管理者は、50 個のコアを QoS **Guaranteed** Pod に割り当て、1 個のコアを **BestEffort** または **Burstable** Pod に割り当てます。これは、分離されたプールの容量を 1 コア超えています。CPU 容量が不十分なため、Pod のスケジューリングが失敗します。

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

- レイテンシーの影響を受けやすいワークロードがネットワークインターフェイスコントローラー (NIC) などの特定のハードウェアを使用する場合は、分離されたプール内の CPU が、このハードウェアにできるだけ近いことを確認してください。少なくとも、ワークロードを同じ Non-Uniform Memory Access (NUMA) ノードに配置する必要があります。
- 予約済みプールは、すべての割り込みを処理するために使用されます。システムネットワークに依存する場合は、すべての着信パケット割り込みを処理するために、十分なサイズの予約プールを割り当てます。4.15 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

11.4. NODE TUNING OPERATOR を使用した NIC キューの削減

Node Tuning Operator を使用すると、各ネットワークデバイスのネットワークインターフェイスコントローラー (NIC) のキュー数を調整できます。PerformanceProfile を使用すると、キューの量を予約された CPU の数まで減らすことができます。

11.4.1. パフォーマンスプロファイルによる NIC キューの調整

パフォーマンスプロファイルを使用すると、各ネットワークデバイスのキュー数を調整できます。

サポート対象のネットワークデバイスは以下のとおりです。

- 非仮想ネットワークデバイス
- 複数のキュー (チャネル) をサポートするネットワークデバイス

サポート対象外のネットワークデバイスは以下の通りです。

- Pure Software ネットワークインターフェイス
- ブロックデバイス
- Intel DPDK Virtual Function

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **cluster-admin** 権限を持つユーザーとして、Node Tuning Operator を実行する OpenShift Container Platform クラスターにログインします。
2. お使いのハードウェアとトポロジーに適したパフォーマンスプロファイルを作成して適用します。プロファイルの作成に関するガイダンスは、パフォーマンスプロファイルの作成のセクションを参照してください。
3. この作成したパフォーマンスプロファイルを編集します。

```
$ oc edit -f <your_profile_name>.yaml
```

4. **spec** フィールドに **net** オブジェクトを設定します。オブジェクトリストには、以下の 2 つのフィールドを含めることができます。

   - **userLevelNetworking** は、ブール値フラグとして指定される必須フィールドです。**userLevelNetworking** が **true** の場合、サポートされているすべてのデバイスのキュー数は、予約された CPU 数に設定されます。デフォルトは **false** です。

   - **devices** は、キューを予約 CPU 数に設定するデバイスのリストを指定する任意のフィールドです。デバイスリストに何も指定しないと、設定がすべてのネットワークデバイスに適用されます。設定は以下のとおりです。

     - **InterfaceName**: このフィールドはインターフェイス名を指定し、正または負のシェルスタイルのワイルドカードをサポートします。

       - ワイルドカード構文の例: **<string> .***

       - 負のルールには、感嘆符のプリフィックスが付きます。除外リスト以外のすべてのデバイスにネットキューの変更を適用するには、**!<device>** を使用します (例: **!eno1**)。

     - **vendorID**: 16 ビット (16 進数) として表されるネットワークデバイスベンダー ID。接頭辞は **0x** です。

     - 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: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

6. このパフォーマンスプロファイルの例を使用して、定義されたデバイス識別子に一致するすべてのデバイスの予約 CPU 数にキュー数を設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - interfaceName: "eth1"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

7. このパフォーマンスプロファイルの例を使用して、インターフェイス名 **eth** で始まるすべてのデバイスの予約 CPU 数にキュー数を設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth*"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

8. このパフォーマンスプロファイルの例を使用して、**eno1** 以外の名前のインターフェイスを持つすべてのデバイスの予約 CPU 数にキュー数を設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "!eno1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

9. このパフォーマンスプロファイルの例を使用して、インターフェイス名 **eth0**、**0x1af4** の **vendorID**、および **0x1000** の **deviceID** を持つすべてのデバイスの予約 CPU 数にキュー数を設定します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  cpu:
    isolated: 3-51,55-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - vendorID: "0x1af4"
      deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

10. 更新されたパフォーマンスプロファイルを適用します。

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

関連情報

- パフォーマンスプロファイルの作成。

11.4.2. キューステータスの確認

このセクションでは、さまざまなパフォーマンスプロファイルについて、変更の適用を検証する方法を複数例示しています。

例 1

この例では、サポートされている **すべて** のデバイスの net キュー数は、予約された CPU 数 (2) に設定されます。

パフォーマンスプロファイルの関連セクションは次のとおりです。

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1  #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
# ...
```

- 以下のコマンドを使用して、デバイスに関連付けられたキューのステータスを表示します。

注記

パフォーマンスプロファイルが適用されたノードで、以下のコマンドを実行します。

```
$ ethtool -l <device>
```

- プロファイルの適用前にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:         0
TX:         0
Other:      0
Combined:   4
Current hardware settings:
RX:         0
TX:         0
Other:      0
Combined:   4
```

- プロファイルの適用後にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:         0
TX:         0
Other:      0
Combined:   4
Current hardware settings:
RX:         0
TX:         0
Other:      0
Combined:   2 (1)
```

(1) チャネルを組み合わせると、**すべての** サポート対象のデバイスの予約 CPU の合計数は 2 になります。これは、パフォーマンスプロファイルでの設定内容と一致します。

例 2

この例では、サポートされている **すべて** のネットワークデバイスの net キュー数は、予約された CPU 数 (2) に特定の **vendorID** を指定して、設定されます。

パフォーマンスプロファイルの関連セクションは次のとおりです。

```
apiVersion: performance.openshift.io/v2
metadata:
  name: performance
spec:
  kind: PerformanceProfile
  spec:
    cpu:
      reserved: 0-1  #total = 2
      isolated: 2-8
    net:
      userLevelNetworking: true
      devices:
      - vendorID = 0x1af4
# ...
```

- 以下のコマンドを使用して、デバイスに関連付けられたキューのステータスを表示します。

注記

パフォーマンスプロファイルが適用されたノードで、以下のコマンドを実行します。

```
$ ethtool -l <device>
```

- プロファイルの適用後にキューのステータスを確認します。

```
$ ethtool -l ens4
```

出力例

```
Channel parameters for ens4:
Pre-set maximums:
RX:         0
TX:         0
Other:      0
Combined:   4
Current hardware settings:
RX:         0
TX:         0
Other:      0
Combined:   2 (1)
```

(1) **vendorID=0x1af4** であるサポート対象の全デバイスの合計予約 CPU 数は 2 となります。たとえば、**vendorID=0x1af4** のネットワークデバイス **ens2** が別に存在する場合に、このデバイスも合計で 2 つの net キューを持ちます。これは、パフォーマンスプロファイルでの設定内容と一致します。

例 3

この例では、サポートされている **すべて** のネットワークデバイスが定義したデバイス ID のいずれかに一致する場合に、そのネットワークデバイスの net キュー数は、予約された CPU 数 (2) に設定されます。

udevadm info コマンドで、デバイスの詳細なレポートを確認できます。以下の例では、デバイスは以下のようになります。

```
# 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。

11.4.3. NIC キューの調整に関するロギング

割り当てられたデバイスの詳細を示すログメッセージは、それぞれの Tuned デーモンログに記録されます。以下のメッセージは、**/var/log/tuned/tuned.log** ファイルに記録される場合があります。

- 正常に割り当てられたデバイスの詳細を示す **INFO** メッセージが記録されます。

```
INFO tuned.plugins.base: instance net_test (net): assigning devices ens1, ens2, ens3
```

- 割り当てることのできるデバイスがない場合は、**WARNING** メッセージが記録されます。

```
WARNING  tuned.plugins.base: instance net_test: no matching devices available
```

11.5. 低レイテンシー CNF チューニングステータスのデバッグ

PerformanceProfile カスタムリソース (CR) には、チューニングのステータスを報告し、レイテンシーのパフォーマンスの低下の問題をデバッグするためのステータスフィールドが含まれます。これらのフィールドは、Operator の調整機能の状態を記述する状態について報告します。

パフォーマンスプロファイルに割り当てられるマシン設定プールのステータスが degraded 状態になると典型的な問題が発生する可能性があり、これにより **PerformanceProfile** のステータスが低下します。この場合、マシン設定プールは失敗メッセージを発行します。

Node Tuning Operator には **performanceProfile.spec.status.Conditions** ステータスフィールドが含まれています。

```
Status:
  Conditions:
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                True
    Type:                  Available
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                True
    Type:                  Upgradeable
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                False
    Type:                  Progressing
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                False
    Type:                  Degraded
```

Status フィールドには、 パフォーマンスプロファイルのステータスを示す **Type** 値を指定する **Conditions** が含まれます。

Available

すべてのマシン設定および Tuned プロファイルが正常に作成され、クラスターコンポーネントで利用可能になり、それら (NTO、MCO、Kubelet) を処理します。

Upgradeable

Operator によって維持されるリソースは、アップグレードを実行する際に安全な状態にあるかどうかを示します。

Progressing

パフォーマンスプロファイルからのデプロイメントプロセスが開始されたことを示します。

Degraded

以下の場合にエラーを示します。

- パーマンスプロファイルの検証に失敗しました。
- すべての関連するコンポーネントの作成が完了しませんでした。

これらのタイプには、それぞれ以下のフィールドが含まれます。

Status

特定のタイプの状態 (**true** または **false**)。

Timestamp

トランザクションのタイムスタンプ。

Reason string

マシンの読み取り可能な理由。

Message string

状態とエラーの詳細を説明する人が判読できる理由 (ある場合)。

11.5.1. マシン設定プール

パフォーマンスプロファイルとその作成される製品は、関連付けられたマシン設定プール (MCP) に従ってノードに適用されます。MCP は、カーネル引数、kube 設定、Huge Page の割り当て、および rt-kernel のデプロイメントを含むパフォーマンスプロファイルが作成するマシン設定の適用に関する進捗についての貴重な情報を保持します。パフォーマンスプロファイルコントローラーは MCP の変更を監視し、それに応じてパフォーマンスプロファイルのステータスを更新します。

MCP は、**Degraded** の場合に限りパフォーマンスプロファイルステータスに返し、**performanceProfile.status.condition.Degraded = true** になります。

例

以下の例は、これに作成された関連付けられたマシン設定プール (**worker-cnf**) を持つパフォーマンスプロファイルのサンプルです。

1. 関連付けられたマシン設定プールの状態は degraded (低下) になります。

```
# oc get mcp
```

出力例

```
NAME      CONFIG                                UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
```

```
DEGRADEDMACHINECOUNT   AGE
master       rendered-master-2ee57a93fa6c9181b546ca46e1571d2d       True      False
False      3              3                 3                   0                   2d21h
worker       rendered-worker-d6b2bdc07d9f5a59a6b68950acf25e5f       True      False
False      2              2                 2                   0                   2d21h
worker-cnf   rendered-worker-cnf-6c838641b8a08fff08dbd8b02fb63f7c   False     True
True       2              1                 1                   1                   2d20h
```

2. MCP の **describe** セクションには理由が示されます。

```
# oc describe mcp worker-cnf
```

出力例

```
  Message:               Node node-worker-cnf is reporting: "prepping update:
  machineconfig.machineconfiguration.openshift.io \"rendered-worker-cnf-
40b9996919c08e335f3ff230ce1d170\" not
  found"
    Reason:              1 nodes are reporting degraded status on sync
```

3. degraded (低下) の状態は、**degraded = true** とマークされたパフォーマンスプロファイルの **status** フィールドにも表示されるはずです。

```
# oc describe performanceprofiles performance
```

出力例

```
Message: Machine config pool worker-cnf Degraded Reason: 1 nodes are reporting
degraded status on sync.
Machine config pool worker-cnf Degraded Message: Node yquinn-q8s5v-w-b-
z5lqn.c.openshift-gce-devel.internal is
reporting: "prepping update: machineconfig.machineconfiguration.openshift.io
\"rendered-worker-cnf-40b9996919c08e335f3ff230ce1d170\" not found".    Reason:
MCPDegraded
  Status:  True
  Type:    Degraded
```

11.6. RED HAT サポート向けの低レイテンシーのチューニングデバッグデータの収集

サポートケースを作成する際、ご使用のクラスターについてのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

must-gather ツールを使用すると、ノードのチューニング、NUMA トポロジー、および低レイテンシーの設定に関する問題のデバッグに必要な OpenShift Container Platform クラスターについての診断情報を収集できます。

迅速なサポートを得るには、OpenShift Container Platform と低レイテンシーチューニングの両方の診断情報を提供してください。

11.6.1. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- 監査ログ
- サービスログ

--image 引数を指定してコマンドを実行する際にイメージを指定できます。イメージを指定する際、ツールはその機能または製品に関連するデータを収集します。**oc adm must-gather** を実行すると、新しい Pod がクラスターに作成されます。データは Pod で収集され、**must-gather.local** で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

11.6.2. 低遅延チューニングデータの収集

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、以下を始めとする低レイテンシーチューニングに関連する機能およびオブジェクトが含まれます。

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

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. 次のコマンドを実行してデバッグ情報を収集します。

```
$ oc adm must-gather
```

出力例

```
[must-gather      ] OUT Using must-gather plug-in image: quay.io/openshift-release
When opening a support case, bugzilla, or issue please include the following summary data
along with any other requested information:
ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6
ClusterVersion: Stable at "<cluster_version>"
ClusterOperators:
 All healthy and stable
```

```
[must-gather      ] OUT namespace/openshift-must-gather-8fh4x created
[must-gather      ] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-rhlgc created
[must-gather-5564g] POD 2023-07-17T10:17:37.610340849Z Gathering data for
ns/openshift-cluster-version...
[must-gather-5564g] POD 2023-07-17T10:17:38.786591298Z Gathering data for ns/default...
[must-gather-5564g] POD 2023-07-17T10:17:39.117418660Z Gathering data for
ns/openshift...
[must-gather-5564g] POD 2023-07-17T10:17:39.447592859Z Gathering data for ns/kube-
system...
[must-gather-5564g] POD 2023-07-17T10:17:39.803381143Z Gathering data for
ns/openshift-etcd...

...

Reprinting Cluster State:
When opening a support case, bugzilla, or issue please include the following summary data
along with any other requested information:
ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6
ClusterVersion: Stable at "<cluster_version>"
ClusterOperators:
 All healthy and stable
```

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、**must-gather** ツールによって作成されたディレクトリー名に置き換えます。

注記

圧縮ファイルを作成して、サポートケースにデータを添付したり、パフォーマンスプロファイルの作成時に Performance Profile Creator ラッパースクリプトで使用したりできます。

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

- **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。
- MachineConfig および KubeletConfig の詳細は、ノードの管理 を参照してください。
- Node Tuning Operator の詳細は、ノードチューニング Operator について を参照してください。
- PerformanceProfile の詳細は、Huge Page の設定 を参照してください。
- コンテナーからの Huge Page の消費に関する詳細は、How huge pages are consumed by apps を参照してください。

第12章 プラットフォーム検証のためのレイテンシーテストの実行

Cloud-native Network Functions (CNF) テストイメージを使用して、CNF ワークロードの実行に必要なすべてのコンポーネントがインストールされている CNF 対応の OpenShift Container Platform クラスターでレイテンシーテストを実行できます。レイテンシーテストを実行して、ワークロードのノードチューニングを検証します。

cnf-tests コンテナーイメージは、**registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15** で入手できます。

12.1. レイテンシーテストを実行するための前提条件

レイテンシーテストを実行するには、クラスターが次の要件を満たしている必要があります。

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

関連情報

- クラスターパフォーマンスプロファイルの作成の詳細は、リアルタイム機能を使用したワーカーのプロビジョニング を参照してください。

12.2. レイテンシーの測定

cnf-tests イメージは、3 つのツールを使用してシステムのレイテンシーを測定します。

- **hwlatdetect**
- **cyclictest**
- **oslat**

各ツールには特定の用途があります。信頼できるテスト結果を得るために、ツールを順番に使用します。

hwlatdetect
: ベアメタルハードウェアが達成できるベースラインを測定します。次のレイテンシーテストに進む前に、**hwlatdetect** によって報告されるレイテンシーが必要なしきい値を満たしていることを確認してください。これは、オペレーティングシステムのチューニングによってハードウェアレイテンシーのスパイクを修正することはできないためです。

cyclictest
: **hwlatdetect** が検証に合格した後、リアルタイムのカーネルスケジューラーのレイテンシーを検証します。**cyclictest** ツールは繰り返しタイマーをスケジュールし、希望のトリガー時間と実際のトリガーの時間の違いを測定します。この違いは、割り込みまたはプロセスの優先度によって生じるチューニングで、基本的な問題を発見できます。ツールはリアルタイムカーネルで実行する必要があります。

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

テストでは、次の環境変数が導入されます。

表12.1 レイテンシーテスト環境変数

環境変数	説明
LATENCY_TEST_DELAY	テストの実行を開始するまでの時間を秒単位で指定します。この変数を使用すると、CPU マネージャーの調整ループでデフォルトの CPU プールを更新できるようになります。デフォルト値は 0 です。
LATENCY_TEST_CPUS	レイテンシーテストを実行する Pod が使用する CPU の数を指定します。変数を設定しない場合、デフォルト設定にはすべての分離された CPU が含まれます。
LATENCY_TEST_RUNTIME	レイテンシーテストを実行する必要がある時間を秒単位で指定します。デフォルト値は 300 秒です。 **注記** レイテンシーテストが完了する前に Ginkgo 2.0 テストスイートがタイムアウトしないようにするには、**-ginkgo.timeout** フラグを **LATENCY_TEST_RUNTIME** + 2 分より大きい値に設定します。**LATENCY_TEST_DELAY** 値も設定する場合は、**-ginkgo.timeout** を **LATENCY_TEST_RUNTIME** + **LATENCY_TEST_DELAY** + 2 分より大きい値に設定する必要があります。Ginkgo 2.0 テストスイートのデフォルトのタイムアウト値は 1 時間です。
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。

注記

レイテンシーツールに固有の変数は、統合された変数よりも優先されます。たとえば、**OSLAT_MAXIMUM_LATENCY** が 30 マイクロ秒に設定され、**MAXIMUM_LATENCY** が 10 マイクロ秒に設定されている場合、 **oslat** テストは 30 マイクロ秒の最大許容遅延で実行されます。

12.3. 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。

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_RUNTIME=<time_in_seconds>\
-e MAXIMUM_LATENCY=<time_in_microseconds> \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh \
--ginkgo.v --ginkgo.timeout="24h"
```

3. オプション: **--ginkgo.dryRun** フラグを追加して、レイテンシーテストをドライランモードで実行します。これは、テストでどのようなコマンドが実行されるかを確認するのに役立ちます。

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

5. オプション: **--ginkgo.timeout="24h"** フラグを追加して、レイテンシーテストが完了する前に Ginkgo 2.0 テストスイートがタイムアウトしないようにします。

重要

各テストのデフォルトの実行時間は 300 秒です。有効なレイテンシーテスト結果を得るには、**LATENCY_TEST_RUNTIME** 変数を更新してテストを少なくとも 12 時間実行してください。

12.3.1. hwlatdetect の実行

hwlatdetect ツールは、Red Hat Enterprise Linux (RHEL) 9.x の通常のサブスクリプションを含む **rt-kernel** パッケージで利用できます。

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。

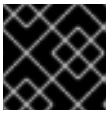
手順

- **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_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="hwlatdetect" --ginkgo.v --ginkgo.timeout="24h"
```

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/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0908 15:25:20.023712      27 request.go:601] Waited for 1.046586367s due to client-side
throttling, not priority and fairness, request:
GET:https://api.hlxcl6.lab.eng.tlv2.redhat.com:6443/apis/imageregistry.operator.openshift.io/v1?
timeout=32s
Running Suite: CNF Features e2e integration tests
==================================================
Random Seed: 1662650718
Will run 1 of 3 specs

[...]

• Failure [283.574 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the hwlatdetect image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:228
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
```

```
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:236

    Log file created at: 2022/09/08 15:25:27
    Running on machine: hwlatdetect-b6n4n
    Binary: Built with gc go1.17.12 for linux/amd64
    Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
    I0908 15:25:27.160620       1 node.go:39] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd1,gpt3)/ostree/rhcos-
c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625769ef5fb9/vmlinuz-4.18.0-
372.19.1.el8_6.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ignition.platform.id=metal
ostree=/ostree/boot.1/rhcos/c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625
769ef5fb9/0 ip=dhcp root=UUID=5f80c283-f6e6-4a27-9b47-a287157483b2 rw
rootflags=prjquota boot=UUID=773bf59a-bafd-48fc-9a87-f62252d739d3 skew_tick=1
nohz=on rcu_nocbs=0-3 tuned.non_isolcpus=0000ffff,ffffffff,fffffff0
systemd.cpu_affinity=4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79 intel_iommu=on iommu=pt
isolcpus=managed_irq,0-3 nohz_full=0-3 tsc=nowatchdog nosoftlockup nmi_watchdog=0
mce=off skew_tick=1 rcutree.kthread_prio=11 + +
    I0908 15:25:27.160830       1 node.go:46] Environment information: kernel version 4.18.0-
372.19.1.el8_6.x86_64
    I0908 15:25:27.160857       1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 100 --window
10000000us --width 950000us]
    F0908 15:27:10.603523       1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect:  test duration 100 seconds
       detector: tracer
       parameters:
            Latency threshold: 1us (1)
            Sample window:     10000000us
            Sample width:      950000us
          Non-sampling period:  9050000us
            Output File:       None

    Starting test
    test finished
    Max Latency: 326us (2)
    Samples recorded: 5
    Samples exceeding threshold: 5
    ts: 1662650739.017274507, inner:6, outer:6
    ts: 1662650749.257272414, inner:14, outer:326
    ts: 1662650779.977272835, inner:314, outer:12
    ts: 1662650800.457272384, inner:3, outer:9
    ts: 1662650810.697273520, inner:3, outer:2

[...]

JUnit report was created: /junit.xml/cnftests-junit.xml


Summarizing 1 Failure:

[Fail] [performance] Latency Test with the hwlatdetect image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:476
```

```
Ran 1 of 194 Specs in 365.797 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (366.08s)
FAIL
```

1. **MAXIMUM_LATENCY**または**HWLATDETECT_MAXIMUM_LATENCY**環境変数を使用して、レイテンシーしきい値を設定できます。
2. テスト中に測定される最大レイテンシー値。

hwlatdetect テスト結果の例
以下のタイプの結果をキャプチャーできます。

- テスト中に行われた変更への影響の履歴を作成するために、各実行後に収集される大まかな結果
- 最良の結果と設定を備えたラフテストの組み合わせセット

良い結果の例

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:
Latency threshold: 10us
Sample window: 1000000us
Sample width: 950000us
Non-sampling period: 50000us
Output File: None

Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
```

hwlatdetect ツールは、サンプルが指定されたしきい値を超えた場合にのみ出力を提供します。

悪い結果の例

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:Latency threshold: 10usSample window: 1000000us
Sample width: 950000usNon-sampling period: 50000usOutput File: None

Starting tests:1610542421.275784439, inner:78, outer:81
ts: 1610542444.330561619, inner:27, outer:28
ts: 1610542445.332549975, inner:39, outer:38
ts: 1610542541.568546097, inner:47, outer:32
ts: 1610542590.681548531, inner:13, outer:17
ts: 1610543033.818801482, inner:29, outer:30
ts: 1610543080.938801990, inner:90, outer:76
ts: 1610543129.065549639, inner:28, outer:39
ts: 1610543474.859552115, inner:28, outer:35
ts: 1610543523.973856571, inner:52, outer:49
```

```
ts: 1610543572.089799738, inner:27, outer:30
ts: 1610543573.091550771, inner:34, outer:28
ts: 1610543574.093555202, inner:116, outer:63
```

hwlatdetect の出力は、複数のサンプルがしきい値を超えていることを示しています。ただし、同じ出力は、次の要因に基づいて異なる結果を示す可能性があります。

- テストの期間
- CPU コアの数
- ホストファームウェアの設定

警告

次のレイテンシーテストに進む前に、**hwlatdetect** によって報告されたレイテンシーが必要なしきい値を満たしていることを確認してください。ハードウェアによって生じるレイテンシーを修正するには、システムベンダーのサポートに連絡しないといけない場合があります。

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

12.3.2. 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。

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。
- Node Tuning Operator を使用してクラスターパフォーマンスプロファイルを適用しました。

手順

- **cyclictest** を実行するには、次のコマンドを実行し、必要に応じて変数の値を置き換えます。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
```

```
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="cyclictest" --ginkgo.v --ginkgo.timeout="24h"
```

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、一般的に {rds} ワークロードでは受け入れられません。

結果がレイテンシーのしきい値を超えると、テストは失敗します。

重要

有効な結果を得るには、テストを少なくとも 12 時間実行する必要があります。

障害出力の例

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=cyclictest
I0908 13:01:59.193776      27 request.go:601] Waited for 1.046228824s due to client-side
throttling, not priority and fairness, request: GET:https://api.compute-
1.example.com:6443/apis/packages.operators.coreos.com/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=================================================
Random Seed: 1662642118
Will run 1 of 3 specs

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the cyclictest image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:220

Ran 1 of 194 Specs in 161.151 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.48s)
FAIL
```

サイクルテスト結果の例
同じ出力は、ワークロードごとに異なる結果を示す可能性があります。たとえば、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
# Avg Latencies: 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:
```

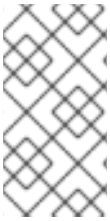
悪い結果の例

```
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
000001 000000   000000  000000  000000  000000  000000  000000  000000  000000  000000
000000  000000  000000  000000  000000  000000
000002 564632   579686  354911  563036  492543  521983  515884  378266  592621  463547
482764  591976  590409  588145  589556  353518
More histogram entries ...
# Total: 000599999 000599999 000599999 000599997 000599997 000599998 000599998
000599997 000599997 000599996 000599995 000599996 000599995 000599995 000599995
000599993
# Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
# Max Latencies: 00493 00387 00271 00619 00541 00513 00009 00389 00252 00215 00539 00498
00363 00204 00068 00520
# Histogram Overflows: 00001 00001 00001 00002 00002 00001 00000 00001 00001 00001 00002
00001 00001 00001 00001 00002
# Histogram Overflow at cycle number:
# Thread 0: 155922
```

```
# Thread 1: 110064
# Thread 2: 110064
# Thread 3: 110063 155921
# Thread 4: 110063 155921
# Thread 5: 155920
# Thread 6:
# Thread 7: 110062
# Thread 8: 110062
# Thread 9: 155919
# Thread 10: 110061 155919
# Thread 11: 155918
# Thread 12: 155918
# Thread 13: 110060
# Thread 14: 110060
# Thread 15: 110059 155917
```

12.3.3. 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。



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。
- Node Tuning Operator を使用してクラスターパフォーマンスプロファイルを適用しました。

手順

- **oslat** テストを実行するには、変数値を適切に置き換えて、次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.focus="oslat" --ginkgo.v --ginkgo.timeout="24h"
```

LATENCY_TEST_CPUS は、**oslat** コマンドでテストする CPU の数を指定します。

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/cnftests -ginkgo.v -ginkgo.focus=oslat
I0908 12:51:55.999393      27 request.go:601] Waited for 1.044848101s due to client-side
throttling, not priority and fairness, request: GET:https://compute-
1.example.com:6443/apis/machineconfiguration.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
==================================================
Random Seed: 1662641514
Will run 1 of 3 specs

[...]

• Failure [77.833 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
  with the oslat image
  /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:128
    should succeed [It]
    /remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:153

    The current latency 304 is bigger than the expected one 1 : 1

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the oslat image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:177

Ran 1 of 194 Specs in 161.091 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.42s)
FAIL
```

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。

12.4. 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 registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --report <report_folder_path> --ginkgo.v
```

ここでは、以下のようになります。

<report_folder_path>
レポートが生成されるフォルダーへのパスです。

12.5. 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。

注記

このコマンドを実行する前に、**junit** フォルダーを作成する必要があります。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/junit:/junit \
-e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.junit-report junit/<file-name>.xml --ginkgo.v
```

ここでは、以下のようになります。

junit
junit レポートを保存するフォルダーです。

12.6. 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。

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。

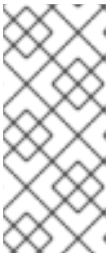
前提条件

- 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。
- Node Tuning Operator を使用してクラスターパフォーマンスプロファイルを適用しました。

手順

- 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 LATENCY_TEST_RUNTIME=<time_in_seconds> registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 \
/usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```



注記

各テストのデフォルトの実行時間は 300 秒です。有効なレイテンシーテスト結果を得るには、**LATENCY_TEST_RUNTIME** 変数を更新してテストを少なくとも 12 時間実行してください。バケットのレイテンシー検証ステップを実行するには、最大レイテンシーを指定する必要があります。最大レイテンシー変数の詳細については、「レイテンシーの測定」セクションの表を参照してください。

テストスイートの実行後に、未解決のリソースすべてがクリーンアップされます。

12.7. 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:v4.15 \
/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 IMAGE_REGISTRY="<disconnected_registry>" \
-e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.15" \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
<disconnected_registry>/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh --ginkgo.v --
ginkgo.timeout="24h"
```

カスタムレジストリーからのイメージを使用するためのテストの設定

CNF_TESTS_IMAGE 変数と **IMAGE_REGISTRY** 変数を使用して、カスタムテストイメージとイメージレジストリーを使用してレイテンシーテストを実行できます。

- カスタムテストイメージとイメージレジストリーを使用するようにレイテンシーテストを設定するには、次のコマンドを実行します。

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY="<custom_image_registry>" \
-e CNF_TESTS_IMAGE="<custom_cnf-tests_image>" \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.15 /usr/bin/test-run.sh --ginkgo.v --
ginkgo.timeout="24h"
```

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-rhel8:4.15 \
/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 LATENCY_TEST_RUNTIME=<time_in_seconds> \
-e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests cnf-tests-local:latest /usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```

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:4.15"
    }
]
```

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:v4.15 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubeconfig/images.json" \
| oc image mirror -f -
```

12.8. 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:v4.15 \
oc get nodes
```

このコマンドが機能しない場合は、DNS 間のスパン、MTU サイズ、またはファイアウォールアクセスに関連するエラーが発生している可能性があります。

第13章 ワーカーレイテンシープロファイルを使用したレイテンシーの高い環境でのクラスターの安定性の向上

クラスター管理者が遅延テストを実行してプラットフォームを検証した際に、遅延が大きい場合でも安定性を確保するために、クラスターの動作を調整する必要性が判明することがあります。クラスター管理者が変更する必要があるのは、ファイルに記録されている 1 つのパラメーターだけです。このパラメーターは、監視プロセスがステータスを読み取り、クラスターの健全性を解釈する方法に影響を与える 4 つのパラメーターを制御するものです。1 つのパラメーターのみを変更し、サポートしやすく簡単な方法でクラスターをチューニングできます。

Kubelet プロセスは、クラスターの健全性を監視する上での出発点です。**Kubelet** は、OpenShift Container Platform クラスター内のすべてのノードのステータス値を設定します。Kubernetes コントローラーマネージャー (**kube controller**) は、デフォルトで 10 秒ごとにステータス値を読み取ります。ノードのステータス値を読み取ることができない場合、設定期間が経過すると、**kube controller** とそのノードとの接続が失われます。デフォルトの動作は次のとおりです。

1. コントロールプレーン上のノードコントローラーが、ノードの健全性を **Unhealthy** に更新し、ノードの **Ready** 状態を `Unknown` とマークします。

2. この操作に応じて、スケジューラーはそのノードへの Pod のスケジューリングを停止します。

3. ノードライフサイクルコントローラーが、**NoExecute** effect を持つ **node.kubernetes.io/unreachable** テイントをノードに追加し、デフォルトでノード上のすべての Pod を 5 分後にエビクトするようにスケジュールします。

この動作は、ネットワークが遅延の問題を起こしやすい場合、特にネットワークエッジにノードがある場合に問題が発生する可能性があります。場合によっては、ネットワークの遅延が原因で、Kubernetes コントローラーマネージャーが正常なノードから更新を受信できないことがあります。**Kubelet** は、ノードが正常であっても、ノードから Pod を削除します。

この問題を回避するには、**ワーカーレイテンシープロファイル** を使用して、**Kubelet** と Kubernetes コントローラーマネージャーがアクションを実行する前にステータスの更新を待機する頻度を調整できます。これらの調整により、コントロールプレーンとワーカーノード間のネットワーク遅延が最適でない場合に、クラスターが適切に動作するようになります。

これらのワーカーレイテンシープロファイルには、3 つのパラメーターセットが含まれています。パラメーターは、遅延の増加に対するクラスターの反応を制御するように、慎重に調整された値で事前定義されています。試験により手作業で最良の値を見つける必要はありません。

クラスターのインストール時、またはクラスターネットワークのレイテンシーの増加に気付いたときはいつでも、ワーカーレイテンシープロファイルを設定できます。

13.1. ワーカーレイテンシープロファイルについて

ワーカーレイテンシープロファイルは、4 つの異なるカテゴリーからなる慎重に調整されたパラメーターです。これらの値を実装する 4 つのパラメーターは、**node-status-update-frequency**、**node-monitor-grace-period**、**default-not-ready-toleration-seconds**、および **default-unreachable-toleration-seconds** です。これらのパラメーターにより、遅延の問題に対するクラスターの反応を制御できる値を使用できます。手作業で最適な値を決定する必要はありません。

重要

これらのパラメーターの手動設定はサポートされていません。パラメーター設定が正しくないと、クラスターの安定性に悪影響が及びます。

すべてのワーカーレイテンシープロファイルは、次のパラメーターを設定します。

node-status-update-frequency

kubelet がノードのステータスを API サーバーにポストする頻度を指定します。

node-monitor-grace-period

Kubernetes コントローラーマネージャーが、ノードを異常とマークし、**node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントをノードに追加する前に、kubelet からの更新を待機する時間を秒単位で指定します。

default-not-ready-toleration-seconds

ノードを異常とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

default-unreachable-toleration-seconds

ノードを到達不能とマークした後、Kube API Server Operator がそのノードから Pod を削除するまでに待機する時間を秒単位で指定します。

次の Operator は、ワーカーレイテンシープロファイルの変更を監視し、それに応じて対応します。

- Machine Config Operator (MCO) は、ワーカーノードの **node-status-update-frequency** パラメーターを更新します。
- Kubernetes コントローラーマネージャーは、コントロールプレーンノードの **node-monitor-grace-period** パラメーターを更新します。
- Kubernetes API Server Operator は、コントロールプレーンノードの **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** パラメーターを更新します。

ほとんどの場合はデフォルト設定が機能しますが、OpenShift Container Platform は、ネットワークで通常よりも高いレイテンシーが発生している状況に対して、他に 2 つのワーカーレイテンシープロファイルを提供します。次のセクションでは、3 つのワーカーレイテンシープロファイルについて説明します。

デフォルトのワーカーレイテンシープロファイル

Default プロファイルを使用すると、各 **Kubelet** が 10 秒ごとにステータスを更新します (**node-status-update-frequency**)。**Kube Controller Manager** は、**Kubelet** のステータスを 5 秒ごとにチェックします (**node-monitor-grace-period**)。
Kubernetes コントローラーマネージャーは、**Kubelet** が異常であると判断するまでに、**Kubelet** からのステータス更新を 40 秒待機します。ステータスが提供されない場合、Kubernetes コントローラーマネージャーは、ノードに **node.kubernetes.io/not-ready** または **node.kubernetes.io/unreachable** テイントのマークを付け、そのノードの Pod を削除します。

そのノードの Pod に **NoExecute** テイントがある場合、その Pod は **tolerationSeconds** に従って実行されます。Pod にテイントがない場合、その Pod は 300 秒以内に削除されます (**Kube API Server** の **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** 設定)。

プロファイル	コンポーネント	パラメーター	値
デフォルト	kubelet	**node-status-update-frequency**	10s

プロファイル	コンポーネント	パラメーター	値
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	40s
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

中規模のワーカーレイテンシープロファイル

ネットワークレイテンシーが通常の場合、**MediumUpdateAverageReaction** プロファイルを使用します。

MediumUpdateAverageReaction プロファイルは、kubelet の更新の頻度を 20 秒に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する期間を 2 分に変更します。そのノード上の Pod の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラーマネージャーは、ノードが異常であると判断するまでに 2 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

ワーカーの低レイテンシープロファイル

ネットワーク遅延が非常に高い場合は、**LowUpdateSlowReaction** プロファイルを使用します。

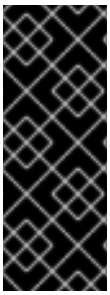
LowUpdateSlowReaction プロファイルは、kubelet の更新頻度を 1 分に減らし、Kubernetes コントローラーマネージャーがそれらの更新を待機する時間を 5 分に変更します。そのノード上の Pod

の Pod 排除期間は 60 秒に短縮されます。Pod に **tolerationSeconds** パラメーターがある場合、エビクションはそのパラメーターで指定された期間待機します。

Kubernetes コントローラーマネージャーは、ノードが異常であると判断するまでに 5 分間待機します。別の 1 分間でエビクションプロセスが開始されます。

プロファイル	コンポーネント	パラメーター	値
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	Kubelet コントローラーマネージャー	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

13.2. クラスター作成時にワーカー遅延プロファイルを実装する



重要

インストーラーの設定を編集するには、まず **openshift-install create manifests** コマンドを使用して、デフォルトのノードマニフェストと他のマニフェスト YAML ファイルを作成する必要があります。このファイル構造を作成しなければ、workerLatencyProfile は追加できません。インストール先のプラットフォームにはさまざまな要件がある場合があります。該当するプラットフォームのドキュメントで、インストールセクションを参照してください。

workerLatencyProfile は、次の順序でマニフェストに追加する必要があります。

1. インストールに適したフォルダー名を使用して、クラスターの構築に必要なマニフェストを作成します。
2. **config.node** を定義する YAML ファイルを作成します。ファイルは **manifests** ディレクトリーに置く必要があります。
3. 初めてマニフェストで **workLatencyProfile** を定義する際には、クラスターの作成時に **Default**、**MediumUpdateAverageReaction**、または **LowUpdateSlowReaction** マニフェストのいずれかを指定します。

検証

- 以下は、マニフェストファイル内の **spec.workerLatencyProfile Default** 値を示すマニフェストを作成する例です。

```
$ openshift-install create manifests --dir=<cluster-install-dir>
```

- マニフェストを編集して値を追加します。この例では、**vi** を使用して、"Default" の **workerLatencyProfile** 値が追加されたマニフェストファイルの例を示します。

```
$ vi <cluster-install-dir>/manifests/config-node-default-profile.yaml
```

出力例

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
name: cluster
spec:
workerLatencyProfile: "Default"
```

13.3. ワーカーレイテンシープロファイルの使用と変更

ネットワークの遅延に対処するためにワーカー遅延プロファイルを変更するには、**node.config** オブジェクトを編集してプロファイルの名前を追加します。遅延が増加または減少したときに、いつでもプロファイルを変更できます。

ワーカーレイテンシープロファイルは、一度に 1 つずつ移行する必要があります。たとえば、**Default** プロファイルから **LowUpdateSlowReaction** ワーカーレイテンシープロファイルに直接移行することはできません。まず **Default** ワーカーレイテンシープロファイルから **MediumUpdateAverageReaction** プロファイルに移行し、次に **LowUpdateSlowReaction** プロファイルに移行する必要があります。同様に、**Default** プロファイルに戻る場合は、まずロープロファイルからミディアムプロファイルに移行し、次に **Default** に移行する必要があります。

注記

OpenShift Container Platform クラスターのインストール時にワーカーレイテンシープロファイルを設定することもできます。

手順

デフォルトのワーカーレイテンシープロファイルから移動するには、以下を実行します。

1. 中規模のワーカーのレイテンシープロファイルに移動します。
   a. **node.config** オブジェクトを編集します。

```
$ oc edit nodes.config/cluster
```

   b. **spec.workerLatencyProfile: MediumUpdateAverageReaction** を追加します。

   node.config オブジェクトの例

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
```

```
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction (1)

# ...
```

(1) 中規模のワーカーレイテンシーポリシーを指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

2. 必要に応じて、ワーカーのレイテンシーが低いプロファイルに移動します。

   a. **node.config** オブジェクトを編集します。

   ```
   $ oc edit nodes.config/cluster
   ```

   b. **spec.workerLatencyProfile** の値を **LowUpdateSlowReaction** に変更します。

   node.config オブジェクトの例

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction (1)

   # ...
   ```

1 低ワーカーレイテンシーポリシーの使用を指定します。

変更が適用されると、各ワーカーノードでのスケジューリングは無効になります。

検証

- 全ノードが **Ready** 状態に戻ると、以下のコマンドを使用して Kubernetes Controller Manager を確認し、これが適用されていることを確認できます。

```
$ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
```

出力例

```
# ...
    - lastTransitionTime: "2022-07-11T19:47:10Z"
      reason: ProfileUpdated
      status: "False"
      type: WorkerLatencyProfileProgressing
    - lastTransitionTime: "2022-07-11T19:47:10Z" 1
      message: all static pod revision(s) have updated latency profile
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
# ...
```

1 プロファイルが適用され、アクティブであることを指定します。

ミディアムプロファイルからデフォルト、またはデフォルトからミディアムに変更する場合、**node.config** オブジェクトを編集し、**spec.workerLatencyProfile** パラメーターを適切な値に設定します。

13.4. WORKERLATENCYPROFILE の結果の値を表示する手順の例

次のコマンドを使用して、**workerLatencyProfile** の値を表示できます。

検証

1. Kube API サーバーによる **default-not-ready-toleration-seconds** および **default-unreachable-toleration-seconds** フィールドの出力を確認します。

```
$ oc get KubeAPIServer -o yaml | grep -A 1 default-
```

出力例

```
default-not-ready-toleration-seconds:
```

```
- "300"
default-unreachable-toleration-seconds:
- "300"
```

2. Kube Controller Manager からの **node-monitor-grace-period** フィールドの値を確認します。

```
$ oc get KubeControllerManager -o yaml | grep -A 1 node-monitor
```

出力例

```
node-monitor-grace-period:
- 40s
```

3. Kubelet からの **nodeStatusUpdateFrequency** 値を確認します。デバッグシェル内のルートディレクトリーとしてディレクトリー **/host** を設定します。root ディレクトリーを **/host** に変更すると、ホストの実行パスに含まれるバイナリーを実行できます。

```
$ oc debug node/<worker-node-name>
$ chroot /host
# cat /etc/kubernetes/kubelet.conf|grep nodeStatusUpdateFrequency
```

出力例

```
  “nodeStatusUpdateFrequency”: “10s”
```

これらの出力は、Worker Latency Profile のタイミング変数のセットを検証します。

第14章 パフォーマンスプロファイルの作成

Performance Profile Creator (PPC) ツールおよび、PPC を使用してパフォーマンスプロファイルを作成する方法を説明します。

注記

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

14.1. PERFORMANCE PROFILE CREATOR の概要

Performance Profile Creator (PPC) は、Node Tuning Operator に付属するコマンドラインツールで、パフォーマンスプロファイルを作成するために使用されます。このツールは、クラスターからの **must-gather** データと、ユーザー指定のプロファイル引数を複数使用します。PPC は、ハードウェアとトポロジーに適したパフォーマンスプロファイルを作成します。

このツールは、以下のいずれかの方法で実行します。

- **podman** の呼び出し
- ラッパースクリプトの呼び出し

14.1.1. must-gather コマンドを使用したクラスターに関するデータの収集

Performance Profile Creator (PPC) ツールには **must-gather** データが必要です。クラスター管理者は、**must-gather** コマンドを実行し、クラスターについての情報を取得します。

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. 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. 次のコマンドを実行してクラスター情報を収集します。

```
$ oc adm must-gather
```

5. オプション: **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/
```

注記

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。

14.1.2. podman を使用した Performance Profile Creator の実行

クラスター管理者は、**podman** および Performance Profile Creator を実行してパフォーマンスプロファイルを作成できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- ベアメタルハードウェアにインストールされたクラスター。
- **podman** および OpenShift CLI (**oc**) がインストールされているノード。
- NodeTuningOperator イメージへのアクセス。

手順

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 --rm --entrypoint performance-profile-creator registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 -h
```

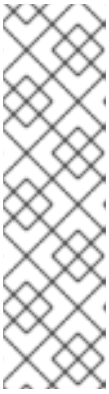
出力例

```
A tool that automates creation of Performance Profiles

Usage:
  performance-profile-creator [flags]

Flags:
      --disable-ht                  Disable Hyperthreading
  -h, --help                        help for performance-profile-creator
      --info string                 Show cluster information; requires --must-gather-dir-path, ignore the other arguments. [Valid values: log, json] (default "log")
      --mcp-name string             MCP name corresponding to the target machines (required)
      --must-gather-dir-path string Must gather directory path (default "must-gather")
      --offlined-cpu-count int      Number of offlined CPUs
      --per-pod-power-management    Enable Per Pod Power Management
      --power-consumption-mode string The power consumption mode.  [Valid values: default, low-latency, ultra-low-latency] (default "default")
      --profile-name string         Name of the performance profile to be created (default "performance")
      --reserved-cpu-count int      Number of reserved CPUs (required)
      --rt-kernel                   Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string Kubelet Topology Manager Policy of the performance profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default "restricted")
      --user-level-networking       Run with User level Networking(DPDK) enabled
```

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 <path_to_must-gather>/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 --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/ose-cluster-node-tuning-operator:v4.15 --mcp-name=worker-cnf
--reserved-cpu-count=4 --rt-kernel=true --split-reserved-cpus-across-numa=false --must-
gather-dir-path /must-gather --power-consumption-mode=ultra-low-latency --offlined-cpu-
count=6 > my-performance-profile.yaml
```

注記

Performance Profile Creator の引数については Performance Profile Creator 引数の表に示しています。必要な引数は、以下の通りです。

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

この例の **mcp-name** 引数は、コマンド **oc get mcp** の出力に基づいて **worker-cnf** に設定されます。シングルノード OpenShift の場合は、**--mcp-name=master** を使用します。

6. 作成した YAML ファイルを確認します。

```
$ cat my-performance-profile.yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 2-39,48-79
    offlined: 42-47
    reserved: 0-1,40-41
```

```
  machineConfigPoolSelector:
    machineconfiguration.openshift.io/role: worker-cnf
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
  workloadHints:
    highPowerConsumption: true
    realTime: true
```

7. 生成されたプロファイルを適用します。

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

14.1.2.1. podman を実行してパフォーマンスプロファイルを作成する方法

以下の例では、**podman** を実行して、NUMA ノード間で分割される、予約済み CPU 20 個を指定してパフォーマンスプロファイルを作成する方法を説明します。

ノードのハードウェア設定:

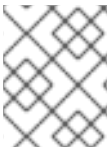
- CPU 80 個
- ハイパースレッディングを有効にする
- NUMA ノード 2 つ
- NUMA ノード 0 に偶数個の CPU、NUMA ノード 1 に奇数個の CPU を稼働させる

podman を実行してパフォーマンスプロファイルを作成します。

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15 --mcp-name=worker-cnf --
reserved-cpu-count=20 --rt-kernel=true --split-reserved-cpus-across-numa=true --must-gather-dir-
path /must-gather > my-performance-profile.yaml
```

作成されたプロファイルは以下の YAML に記述されます。

```
  apiVersion: performance.openshift.io/v2
  kind: PerformanceProfile
  metadata:
    name: performance
  spec:
    cpu:
      isolated: 10-39,50-79
      reserved: 0-9,40-49
    nodeSelector:
      node-role.kubernetes.io/worker-cnf: ""
    numa:
      topologyPolicy: restricted
    realTimeKernel:
      enabled: true
```



注記

この場合、CPU 10 個が NUMA ノード 0 に、残りの 10 個は NUMA ノード 1 に予約されます。

14.1.3. Performance Profile Creator ラッパースクリプトの実行

パフォーマンスプロファイルラッパースクリプトをし用すると、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

- NodeTuningOperator イメージへのアクセス。
- **must-gather** tarball にアクセスできる。

手順

1. ローカルマシンにファイル (例: **run-perf-profile-creator.sh**) を作成します。

```
$ vi run-perf-profile-creator.sh
```

2. ファイルに以下のコードを貼り付けます。

```
#!/bin/bash

readonly CONTAINER_RUNTIME=${CONTAINER_RUNTIME:-podman}
readonly CURRENT_SCRIPT=$(basename "$0")
readonly CMD="${CONTAINER_RUNTIME} run --entrypoint performance-profile-creator"
readonly IMG_EXISTS_CMD="${CONTAINER_RUNTIME} image exists"
readonly IMG_PULL_CMD="${CONTAINER_RUNTIME} image pull"
readonly MUST_GATHER_VOL="/must-gather"

NTO_IMG="registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15"
MG_TARBALL=""
DATA_DIR=""

usage() {
  print "Wrapper usage:"
  print "  ${CURRENT_SCRIPT} [-h] [-p image][-t path] -- [performance-profile-creator flags]"
  print ""
  print "Options:"
  print "   -h              help for ${CURRENT_SCRIPT}"
  print "   -p              Node Tuning Operator image"
  print "   -t              path to a must-gather tarball"

  ${IMG_EXISTS_CMD} "${NTO_IMG}" && ${CMD} "${NTO_IMG}" -h
}

function cleanup {
  [ -d "${DATA_DIR}" ] && rm -rf "${DATA_DIR}"
}
trap cleanup EXIT
```

```
exit_error() {
  print "error: $*"
  usage
  exit 1
}

print() {
  echo  "$*" >&2
}

check_requirements() {
  ${IMG_EXISTS_CMD} "${NTO_IMG}" || ${IMG_PULL_CMD} "${NTO_IMG}" || \
      exit_error "Node Tuning Operator image not found"

  [ -n "${MG_TARBALL}" ] || exit_error "Must-gather tarball file path is mandatory"
  [ -f "${MG_TARBALL}" ] || exit_error "Must-gather tarball file not found"

  DATA_DIR=$(mktemp -d -t "${CURRENT_SCRIPT}XXXX") || exit_error "Cannot create the
data directory"
  tar -zxf "${MG_TARBALL}" --directory "${DATA_DIR}" || exit_error "Cannot decompress the
must-gather tarball"
  chmod a+rx "${DATA_DIR}"

  return 0
}

main() {
  while getopts ':hp:t:' OPT; do
    case "${OPT}" in
      h)
        usage
        exit 0
        ;;
      p)
        NTO_IMG="${OPTARG}"
        ;;
      t)
        MG_TARBALL="${OPTARG}"
        ;;
      ?)
        exit_error "invalid argument: ${OPTARG}"
        ;;
    esac
  done
  shift $((OPTIND - 1))

  check_requirements || exit 1

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

main "$@"
```

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              Node Tuning Operator image 1
   -t              path to a must-gather tarball 2
A tool that automates creation of Performance Profiles

Usage:
  performance-profile-creator [flags]

Flags:
      --disable-ht                  Disable Hyperthreading
  -h, --help                        help for performance-profile-creator
      --info string                 Show cluster information; requires --must-gather-dir-path,
ignore the other arguments. [Valid values: log, json] (default "log")
      --mcp-name string             MCP name corresponding to the target machines
(required)
      --must-gather-dir-path string Must gather directory path (default "must-gather")
      --offlined-cpu-count int      Number of offlined CPUs
      --per-pod-power-management    Enable Per Pod Power Management
      --power-consumption-mode string  The power consumption mode.  [Valid values:
default, low-latency, ultra-low-latency] (default "default")
      --profile-name string         Name of the performance profile to be created (default
"performance")
      --reserved-cpu-count int      Number of reserved CPUs (required)
      --rt-kernel                   Enable Real Time Kernel (required)
      --split-reserved-cpus-across-numa  Split the Reserved CPUs across NUMA nodes
      --topology-manager-policy string  Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
      --user-level-networking       Run with User level Networking(DPDK) enabled
```

注記

引数には、以下の 2 つのタイプがあります。

- ラッパー引数名は、**-h**、**-p**、および **-t** です。
- PPC 引数

1 オプション: Node Tuning Operator のイメージを指定します。設定されていない場合は、デフォルトのアップストリームイメージ (**registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.15**) が使用されます。

2 **-t** は、必須のラッパースクリプトの引数で、**must-gather** tarball へのパスを指定します。

5. Performance Profile Creator ツールを検出モードで実行します。

注記

検出モードは、**must-gather** からの出力を使用してクラスターを検査します。生成された出力には、以下のような情報が含まれます。

- 割り当てられた CPU ID を使用した NUMA セルのパーティション設定
- ハイパースレッディングが有効にされているかどうか

この情報を使用して、Performance Profile Creator ツールにわたす一部の引数に適切な値を設定できます。

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --info=log
```

注記

info オプションでは、出力形式を指定する値が必要です。使用できる値は log と JSON です。JSON 形式はデバッグ用に確保されています。

6. マシン設定プールを確認します。

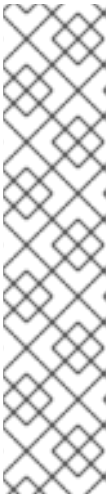
```
$ oc get mcp
```

出力例

```
NAME        CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master      rendered-master-acd1358917e9f98cbdb599aea622d78b      True      False
False      3              3              3                   0                   22h
worker-cnf   rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826   False     True
False      2              1              1                   0                   22h
```

7. パフォーマンスプロファイルを作成します。

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --mcp-name=worker-cnf --
reserved-cpu-count=2 --rt-kernel=true > my-performance-profile.yaml
```



注記

Performance Profile Creator の引数については Performance Profile Creator 引数の表に示しています。必要な引数は、以下の通りです。

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

この例の **mcp-name** 引数は、コマンド **oc get mcp** の出力に基づいて **worker-cnf** に設定されます。シングルノード OpenShift の場合は、 **--mcp-name=master** を使用します。

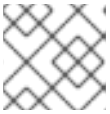
8. 作成した YAML ファイルを確認します。

```
$ cat my-performance-profile.yaml
```

出力例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 1-39,41-79
    reserved: 0,40
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: false
```

9. 生成されたプロファイルを適用します。



注記

プロファイルを適用する前に、Node Tuning Operator をインストールします。

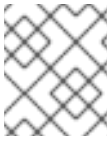
```
$ oc apply -f my-performance-profile.yaml
```

14.1.4. Performance Profile Creator の引数

表14.1 Performance Profile Creator の引数

引数	説明

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。

引数	説明
offlined-cpu-count	オフラインの CPU の数。  **注記** これは 0 より大きい自然数でなければなりません。十分な数の論理プロセッサーがオフラインにされていない場合、エラーメッセージがログに記録されます。メッセージは次のとおりです。 > Error: failed to compute the reserved and isolated CPUs: please ensure that reserved-cpu-count plus offlined-cpu-count should be in the range [0,1] > Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]
power-consumption-mode	電力消費モード。 以下の値を使用できます。 - **default**: 有効な電力管理と基本的な低遅延を備えた CPU パーティション。 - **low-latency**: レイテンシーの数値を改善するための強化された対策。 - **ultra-low-latency**: 電力管理を犠牲にして、最適な遅延を優先します。 デフォルト: **default**。
per-pod-power-management	Pod ごとの電源管理を有効にします。電力消費モードとして **Ultra-low-latency** を設定している場合、この引数は使用できません。 使用できる値は **true** または **false** です。 デフォルト: **false**。
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。

引数	説明
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**。

14.2. パフォーマンスプロファイルの参照

14.2.1. OpenStack で OVS-DPDK を使用するクラスター用のパフォーマンスプロファイルテンプレート

Red Hat OpenStack Platform (RHOSP) で Open vSwitch と Data Plane Development Kit (OVS-DPDK) を使用するクラスターでマシンのパフォーマンスを最大化するには、パフォーマンス プロファイルを使用できます。

次のパフォーマンスプロファイル テンプレートを使用して、デプロイメント用のプロファイルを作成できます。

OVS-DPDK を使用するクラスターのパフォーマンスプロファイル テンプレート

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: cnf-performanceprofile
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
```

```
  - processor.max_cstate=1
  - idle=poll
  - intel_idle.max_cstate=0
  - default_hugepagesz=1GB
  - hugepagesz=1G
  - intel_iommu=on
cpu:
  isolated: <CPU_ISOLATED>
  reserved: <CPU_RESERVED>
hugepages:
  defaultHugepagesSize: 1G
  pages:
    - count: <HUGEPAGES_COUNT>
      node: 0
      size: 1G
nodeSelector:
  node-role.kubernetes.io/worker: ''
realTimeKernel:
  enabled: false
  globallyDisableIrqLoadBalancing: true
```

CPU_ISOLATED キー、**CPU_RESERVED** キー、および **HUGEPAGES_COUNT** キーの設定に適した値を入力します。

パフォーマンスプロファイルを作成および使用する方法については、OpenShift Container Platform ドキュメントのスケーラビリティとパフォーマンスセクションのパフォーマンスプロファイルの作成ページを参照してください。

14.3. 関連情報

- **must-gather** ツールの詳細は、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15章 ワークロードの分割

リソースに制約のある環境では、ワークロードの分割を使用して、OpenShift Container Platform サービス、クラスター管理ワークロード、インフラストラクチャー Pod を分離し、予約済みの CPU セットで実行できます。

クラスター管理に必要な予約済み CPU の最小数は、4 つの CPU ハイパースレッド (HT) です。ワークロード分割では、クラスター管理ワークロードパーティションに含めるために、一連のクラスター管理 Pod と一連の一般的なアドオン Operator に注釈を付けます。これらの Pod は、最低限のサイズの CPU 設定内で正常に動作します。最小クラスター管理 Pod のセット外の追加の Operator またはワークロードでは、追加の CPU をワークロードパーティションに追加する必要があります。

ワークロード分割は、標準の Kubernetes スケジューリング機能を使用して、ユーザーワークロードをプラットフォームワークロードから分離します。

ワークロードの分割には次の変更が必要です。

1. **install-config.yaml** ファイルに、**cpuPartitioningMode** を追加フィールドとして追加します。

```
apiVersion: v1
baseDomain: devcluster.openshift.com
cpuPartitioningMode: AllNodes (1)
compute:
 - architecture: amd64
   hyperthreading: Enabled
   name: worker
   platform: {}
   replicas: 3
controlPlane:
 architecture: amd64
 hyperthreading: Enabled
 name: master
 platform: {}
 replicas: 3
```

(1) インストール時に CPU のパーティション設定用クラスターをセットアップします。デフォルト値は **None** です。

注記

ワークロードの分割は、クラスターのインストール中にのみ有効にできます。インストール後にワークロードパーティショニングを無効にすることはできません。

2. パフォーマンスプロファイルで、**isolated** および **reserved** CPU を指定します。

推奨されるパフォーマンスプロファイル設定

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
  # matches this name: include=openshift-node-performance-
${PerformanceProfile.metadata.name}
```

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

表15.1 シングルノード OpenShift クラスターの PerformanceProfile CR オプション

PerformanceProfile CR フィールド	説明

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

PerformanceProfile CR フィールド	説明
spec.workloadHints	**workloadHints** を使用して、各種ワークロードの最上位フラグのセットを定義します。この例では、クラスターが低レイテンシーかつ高パフォーマンスになるように設定されています。

ワークロードパーティショニングにより、プラットフォーム Pod に拡張された **Management.workload.openshift.io/cores** リソースタイプが導入されます。kubelet は、対応するリソース内のプールに割り当てられた Pod でリソースと CPU リクエストをアドバタイズします。ワークロードの分割が有効になっている場合、スケジューラーは **management.workload.openshift.io/cores** リソースにより、デフォルトの **cpuset** だけでなく、ホストの **cpushares** 容量に基づいて Pod を適切に割り当てることができます。

関連情報

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

第16章 NODE OBSERVABILITY OPERATOR の使用

Node Observability Operator は、コンピュートノードのスクリプトから CRI-O および Kubelet プロファイリングまたはメトリクスを収集して保存します。

Node Observability Operator を使用すると、プロファイリングデータをクエリーして、CRI-O および Kubelet のパフォーマンス傾向を分析できるようになります。カスタムリソース定義の **run** フィールドを使用して、パフォーマンス関連の問題のデバッグと、ネットワークメトリクスの埋め込みスクリプトの実行をサポートします。CRI-O および Kubelet のプロファイリングまたはスクリプトを有効にするには、カスタムリソース定義で **type** フィールドを設定します。

重要

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

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

16.1. NODE OBSERVABILITY OPERATOR のワークフロー

次のワークフローは、Node Observability Operator を使用してプロファイリングデータをクエリーする方法の概要を示しています。

1. Node Observability Operator を OpenShift Container Platform クラスターにインストールします。
2. NodeObservability カスタムリソースを作成して、選択したワーカーノードで CRI-O プロファイリングを有効にします。
3. プロファイリングクエリーを実行して、プロファイリングデータを生成します。

16.2. NODE OBSERVABILITY OPERATOR のインストール

Node Observability Operator は、デフォルトでは OpenShift Container Platform にインストールされていません。OpenShift Container Platform CLI または Web コンソールを使用して、Node Observability Operator をインストールできます。

16.2.1. CLI を使用した Node Observability Operator のインストール

OpenShift CLI(oc) を使用して、Node Observability Operator をインストールできます。

前提条件

- OpenShift CLI (oc) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 次のコマンドを実行して、Node Observability Operator が使用可能であることを確認します。

```
$ oc get packagemanifests -n openshift-marketplace node-observability-operator
```

出力例

```
NAME                          CATALOG               AGE
node-observability-operator   Red Hat Operators     9h
```

2. 次のコマンドを実行して、**node-observability-operator** namespace を作成します。

```
$ oc new-project node-observability-operator
```

3. **OperatorGroup** オブジェクト YAML ファイルを作成します。

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  targetNamespaces: []
EOF
```

4. **Subscription** オブジェクトの YAML ファイルを作成して、namespace を Operator にサブスクライブします。

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-observability-operator
  namespace: node-observability-operator
spec:
  channel: alpha
  name: node-observability-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

検証

1. 次のコマンドを実行して、インストールプラン名を表示します。

```
$ oc -n node-observability-operator get sub node-observability-operator -o yaml | yq
'.status.installplan.name'
```

出力例

```
install-dt54w
```

2. 次のコマンドを実行して、インストールプランのステータスを確認します。

```
$ oc -n node-observability-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

<install_plan_name> は、前のコマンドの出力から取得したインストール計画名です。

出力例

```
COMPLETE
```

3. Node Observability Operator が稼働していることを確認します。

```
$ oc get deploy -n node-observability-operator
```

出力例

```
NAME                                          READY   UP-TO-DATE  AVAILABLE   AGE
node-observability-operator-controller-manager  1/1     1           1           40h
```

16.2.2. Web コンソールを使用した Node Observability Operator のインストール

Node Observability Operator は、OpenShift Container Platform コンソールからインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. 管理者のナビゲーションパネルで、**Operators → OperatorHub** をデプロイメントします。
3. **All items** フィールドに **Node Observability Operator** と入力し、**Node Observability Operator** タイルを選択します。
4. **Install** をクリックします。
5. **Install Operator** ページで、次の設定を設定します。
   a. **Update channel** 領域で、**alpha** をクリックします。
   b. **Installation mode** 領域で、**A specific namespace on the cluster**をクリックします。
   c. **Installed Namespace** リストから、リストから **node-observability-operator** を選択します。
   d. **Update approval** 領域で、**Automatic** を選択します。
   e. **Install** をクリックします。

検証

検証

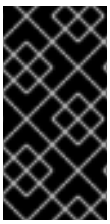
1. 管理者のナビゲーションパネルで、**Operators → Installed Operators** をデプロイメントします。
2. Node Observability Operator が Operators リストにリストされていることを確認します。

16.3. NODE OBSERVABILITY OPERATOR を使用して CRI-O および KUBELET プロファイリングデータをリクエストする

CRI-O および Kubelet プロファイリングデータの収集に使用する、Node Observability カスタムリソースを作成します。

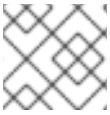
16.3.1. Node Observability カスタムリソースの作成

プロファイリングクエリーを実行する前に、**NodeObservability** カスタムリソース (CR) を作成して実行する必要があります。一致する**NodeObservability** CR を実行すると、必要なマシン設定およびマシン設定プール CR が作成され、**nodeSelector** に一致するワーカーノードで CRI-O プロファイリングを有効にします。



重要

ワーカーノードで CRI-O プロファイリングが有効になっていない場合、**NodeObservabilityMachineConfig** リソースが作成されます。**NodeObservability** CR で指定された **nodeSelector** に一致するワーカーノードが再起動します。完了するまでに 10 分以上かかる場合があります。



注記

Kubelet プロファイリングはデフォルトで有効になっています。

ノードの CRI-Ounix ソケットは、エージェント Pod にマウントされます。これにより、エージェントは CRI-O と通信して pprof 要求を実行できます。同様に、**kubelet-serving-ca** 証明書チェーンはエージェント Pod にマウントされ、エージェントとノードの kubelet エンドポイント間の安全な通信を可能にします。

前提条件

- Node Observability Operator をインストールしました。
- OpenShift CLI (oc) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、OpenShift Container Platform CLI にログインします。

```
$ oc login -u kubeadmin https://<HOSTNAME>:6443
```

2. 次のコマンドを実行して、**node-observability-operator** namespace に切り替えます。

```
$ oc project node-observability-operator
```

3. 次のテキストを含む **nodeobservability.yaml** という名前の CR ファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
  name: cluster (1)
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_hostname> (2)
  type: crio-kubelet
```

(1) クラスターごとに **NodeObservability** CR が 1 つしかないため、名前を **cluster** として指定する必要があります。

(2) Node Observability エージェントをデプロイする必要があるノードを指定します。

4. **NodeObservability** CR を実行します。

```
oc apply -f nodeobservability.yaml
```

出力例

```
nodeobservability.olm.openshift.io/cluster created
```

5. 次のコマンドを実行して、**NodeObservability** CR のステータスを確認します。

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

出力例

```
conditions:
  conditions:
  - lastTransitionTime: "2022-07-05T07:33:54Z"
    message: 'DaemonSet node-observability-ds ready: true NodeObservabilityMachineConfig
      ready: true'
    reason: Ready
    status: "True"
    type: Ready
```

NodeObservability CR の実行は、理由が **Ready** で、ステータスが **True** のときに完了します。

16.3.2. プロファイリングクエリーの実行

プロファイリングクエリーを実行するには、**NodeObservabilityRun** リソースを作成する必要があります。プロファイリングクエリーは、CRI-O および Kubelet プロファイリングデータを 30 秒間フェッチするブロッキング操作です。プロファイリングクエリーが完了したら、コンテナーファイルシステムの **/run/node-observability** ディレクトリー内のプロファイリングデータを取得する必要があります。データの有効期間は、**emptyDir** ボリュームを介してエージェント Pod にバインドされるため、エージェント Pod が **running** の状態にある間にプロファイリングデータにアクセスできます。

重要

一度にリクエストできるプロファイリングクエリーは 1 つだけです。

前提条件

- Node Observability Operator をインストールしました。
- **NodeObservability** カスタムリソース (CR) を作成しました。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 次のテキストを含む **nodeobservabilityrun.yaml** という名前の **NodeObservabilityRun** リソースファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun
spec:
  nodeObservabilityRef:
    name: cluster
```

2. **NodeObservabilityRun** リソースを実行して、プロファイリングクエリーをトリガーします。

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

3. 次のコマンドを実行して、**NodeObservabilityRun** のステータスを確認します。

```
$ oc get nodeobservabilityrun nodeobservabilityrun -o yaml  | yq '.status.conditions'
```

出力例

```
conditions:
- lastTransitionTime: "2022-07-07T14:57:34Z"
  message: Ready to start profiling
  reason: Ready
  status: "True"
  type: Ready
- lastTransitionTime: "2022-07-07T14:58:10Z"
  message: Profiling query done
  reason: Finished
  status: "True"
  type: Finished
```

ステータスが **True** になり、タイプが **Finished** になると、プロファイリングクエリーは完了です。

4. 次の bash スクリプトを実行して、コンテナーの **/run/node-observability** パスからプロファイリングデータを取得します。

```
for a in $(oc get nodeobservabilityrun nodeobservabilityrun -o yaml | yq
```

```
.status.agents[].name); do
  echo "agent ${a}"
  mkdir -p "/tmp/${a}"
  for p in $(oc exec "${a}" -c node-observability-agent -- bash -c "ls /run/node-observability/*.pprof"); do
    f="$(basename ${p})"
    echo "copying ${f} to /tmp/${a}/${f}"
    oc exec "${a}" -c node-observability-agent -- cat "${p}" > "/tmp/${a}/${f}"
  done
done
```

16.4. NODE OBSERVABILITY OPERATOR スクリプト

スクリプトを作成すると、現在の Node Observability Operator および Node Observability Agent を使用して、事前設定された bash スクリプトを実行できます。

これらのスクリプトは、CPU 負荷、メモリーの逼迫、ワーカーノードの問題などの主要なメトリクスを監視します。sar レポートとカスタムパフォーマンスメトリクスも収集します。

16.4.1. スクリプト用のノード監視カスタムリソースを作成する

スクリプトを実行する前に、**NodeObservability** カスタムリソース (CR) を作成して実行する必要があります。**NodeObservability** CR を実行すると、**nodeSelector** ラベルに一致するコンピュートノード上でエージェントがスクリプトモードで有効になります。

前提条件

- Node Observability Operator をインストールしました。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 以下のコマンドを実行して、OpenShift Container Platform クラスターにログインします。

```
$ oc login -u kubeadmin https://<host_name>:6443
```

2. 次のコマンドを実行して、**node-observability-operator** namespace に切り替えます。

```
$ oc project node-observability-operator
```

3. 次の内容を含む **nodeobservability.yaml** という名前のファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
  name: cluster (1)
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_hostname> (2)
  type: scripting (3)
```

1. クラスターごとに **NodeObservability** CR が 1 つしかないため、名前を **cluster** として指定する必要があります。
2. Node Observability エージェントをデプロイする必要があるノードを指定します。
3. エージェントをスクリプトモードでデプロイするには、タイプを **scripting** に設定する必要があります。

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

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

出力例

```
nodeobservability.olm.openshift.io/cluster created
```

5. 次のコマンドを実行して、**NodeObservability** CR のステータスを確認します。

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

出力例

```
conditions:
  conditions:
  - lastTransitionTime: "2022-07-05T07:33:54Z"
    message: 'DaemonSet node-observability-ds ready: true NodeObservabilityScripting
      ready: true'
    reason: Ready
    status: "True"
    type: Ready
```

NodeObservability CR の実行は、**reason** が **Ready**、**status** が **"True"**. の場合に完了します。

16.4.2. Node Observability Operator スクリプトの設定

前提条件

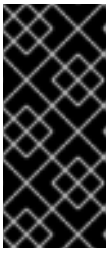
- Node Observability Operator をインストールしました。
- **NodeObservability** カスタムリソース (CR) を作成しました。
- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 次の内容を含む、**nodeobservabilityrun-script.yaml** という名前のファイルを作成します。

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
  name: nodeobservabilityrun-script
  namespace: node-observability-operator
```

```
spec:
  nodeObservabilityRef:
    name: cluster
    type: scripting
```



重要

次のスクリプトのみをリクエストできます。

- **metrics.sh**
- **network-metrics.sh** (**monitor.sh** を使用)

2. 次のコマンドを使用して **NodeObservabilityRun** リソースを作成することで、スクリプトをトリガーします。

```
$ oc apply -f nodeobservabilityrun-script.yaml
```

3. 次のコマンドを実行して、**NodeObservabilityRun** スクリプトのステータスを確認します。

```
$ oc get nodeobservabilityrun nodeobservabilityrun-script -o yaml  | yq '.status.conditions'
```

出力例

```
Status:
  Agents:
    Ip:    10.128.2.252
    Name:  node-observability-agent-n2fpm
    Port:  8443
    Ip:    10.131.0.186
    Name:  node-observability-agent-wcc8p
    Port:  8443
  Conditions:
    Conditions:
      Last Transition Time:  2023-12-19T15:10:51Z
      Message:               Ready to start profiling
      Reason:                Ready
      Status:                True
      Type:                  Ready
      Last Transition Time:  2023-12-19T15:11:01Z
      Message:               Profiling query done
      Reason:                Finished
      Status:                True
      Type:                  Finished
  Finished Timestamp:        2023-12-19T15:11:01Z
  Start Timestamp:           2023-12-19T15:10:51Z
```

Status が **True**、**Type** が **Finished** になると、スクリプトの作成は完了です。

4. 次の bash スクリプトを実行して、コンテナーのルートパスからスクリプトデータを取得します。

```
#!/bin/bash
```

```
RUN=$(oc get nodeobservabilityrun --no-headers | awk '{print $1}')

for a in $(oc get nodeobservabilityruns.nodeobservability.olm.openshift.io/${RUN} -o json | jq
.status.agents[].name); do
  echo "agent ${a}"
  agent=$(echo ${a} | tr -d "\"\`")
  base_dir=$(oc exec "${agent}" -c node-observability-agent -- bash -c "ls -t | grep node-
observability-agent" | head -1)
  echo "${base_dir}"
  mkdir -p "/tmp/${agent}"
  for p in $(oc exec "${agent}" -c node-observability-agent -- bash -c "ls ${base_dir}"); do
    f="/${base_dir}/${p}"
    echo "copying ${f} to /tmp/${agent}/${p}"
    oc exec "${agent}" -c node-observability-agent -- cat ${f} > "/tmp/${agent}/${p}"
  done
done
```

16.5. 関連情報

ワーカーメトリクスの収集方法について、詳細は Red Hat ナレッジベースの記事 を参照してください。