OpenShift Container Platform 4.7

スケーラビリティーおよびパフォーマンス

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

Last Updated: 2023-02-24

OpenShift Container Platform 4.7 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

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 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、クラスターを大きなノード数にスケーリングする際に、以下のプラクティスを適用します。

1.1. 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
  machineCIDR: 10.0.0.0/16
  networkType: OpenShiftSDN
  serviceNetwork:
  - 172.30.0.0/16
```

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

第2章 ホストについての推奨されるプラクティス

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

重要

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

2.1. ノードホストについての推奨プラクティス

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

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

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

重要

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

注記

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

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

```
kubeletConfig:
  podsPerCore: 10
```

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

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

```
kubeletConfig:
  maxPods: 250
```

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

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

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

以下のガイダンスを参照してください。

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

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

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

KubeletConfig CR の例

```
$ oc get kubeletconfig
```

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

KubeletConfig マシン設定を示す例

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

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

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

前提条件

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

   a. マシン設定プールを表示します。

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

   以下に例を示します。

   ```
   $ oc describe machineconfigpool worker
   ```

   出力例

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

   (1) ラベルが追加されると、**labels** の下に表示されます。

   b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

手順

1. これは、選択可能なマシン設定オブジェクトを表示します。

   ```
   $ oc get machineconfig
   ```

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

2. ノードあたりの最大 Pod の現在の値を確認します。

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

以下に例を示します。

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

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

出力例

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

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

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

(1) Machine Config Pool からラベルを入力します。

(2) kubelet 設定を追加します。この例では、**maxPods** を使用してノードあたりの最大 Pod を設定します。

注記

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

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

a. ラベルを使用してワーカーのマシン設定プールを更新します。

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

b. **KubeletConfig** オブジェクトを作成します。

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

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

```
$ oc get kubeletconfig
```

出力例

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

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

4. 変更がノードに適用されていることを確認します。

a. **maxPods** 値が変更されたワーカーノードで確認します。

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

b. **Allocatable** スタンザを見つけます。

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

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

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

5. **KubeletConfig** オブジェクトの変更を確認します。

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

上記のコマンドでは **status: "True"** と **type:Success** が表示されるはずです。

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

2.3. 利用不可のワーカーノードの数の変更

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

手順

1. **worker** マシン設定プールを編集します。

```
$ oc edit machineconfigpool worker
```

2. **maxUnavailable** を必要な値に設定します。

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

重要

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

2.4. コントロールプレーンノードのサイジング

コントロールプレーンノードリソースの要件は、クラスター内のノード数によって異なります。コントロールプレーンノードのサイズについての以下の推奨内容は、テストに重点を置いた場合のコントロールプレーンの密度の結果に基づいています。コントロールプレーンのテストでは、ノード数に応じて各 namespace でクラスター全体に展開される以下のオブジェクトを作成します。

- 12 イメージストリーム
- 3 ビルド設定
- 6 ビルド
- それぞれに 2 つのシークレットをマウントする 2 Pod レプリカのある 1 デプロイメント
- 2 つのシークレットをマウントする 1 Pod レプリカのある 2 デプロイメント
- 先のデプロイメントを参照する 3 つのサービス
- 先のデプロイメントを参照する 3 つのルート
- 10 のシークレット (それらの内の 2 つは先ののデプロイメントでマウントされる)
- 10 の設定マップ (それらの内の 2 つは先のデプロイメントでマウントされる)

ワーカーノードの数	クラスターの負荷 (namespace)	CPU コア数	メモリー (GB)
25	500	4	16
100	1000	8	32
250	4000	16	96

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

重要

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

重要

インストール手法にインストーラーでプロビジョニングされるインフラストラクチャーを使用した場合には、実行中の OpenShift Container Platform 4.7 クラスターでコントロールプレーンノードのサイズを変更できません。その代わりに、ノードの合計数を見積もり、コントロールプレーンノードの推奨サイズをインストール時に使用する必要があります。

重要

この推奨内容は、OpenShiftSDN がネットワークプラグインとして設定されている OpenShift Container Platform クラスターでキャプチャーされるデータポイントに基づくものです。

注記

OpenShift Container Platform 4.7 では、デフォルトで CPU コア (500 ミリコア) の半分がシステムによって予約されます (OpenShift Container Platform 3.11 以前のバージョンと比較)。サイズはこれを考慮に入れて決定されます。

2.4.1. Amazon Web Services (AWS) マスターインスタンスのフレーバーサイズを増やす

クラスター内の AWS マスターノードに過負荷がかかり、マスターノードがより多くのリソースを必要とする場合は、マスターインスタンスのフレーバーサイズを増加させることができます。

注記

AWS マスターインスタンスのフレーバーサイズを増やす前に、etcd をバックアップすることをお勧めします。

前提条件

- AWS に IPI (installer-provisioned infrastructure) または UPI (user-provisioned infrastructure) クラスターがある。

手順

1. AWS コンソールを開き、マスターインスタンスを取得します。
2. 1 つのマスターインスタンスを停止します。
3. 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。
4. 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、**m5.xlarge** を **m5.2xlarge** または **m5.4xlarge** に変更できます。
5. インスタンスをバックアップし、次のマスターインスタンスに対して手順を繰り返します。

関連情報

- etcd のバックアップ

2.5. ETCD についての推奨されるプラクティス

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_debugging_mvcc_db_total_size_in_bytes**、これはデフラグを待機している空き領域を含む、データベースのサイズを示します。

etcd の最適化の詳細については、etcd データの最適化を参照してください。

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。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。低遅延と高スループットの SSD または NVMe ディスクでバックアップされたマシンで 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。

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。

- **etcd_disk_wal_fsync_duration_seconds_bucket** メトリックは、etcd ディスクの fsync 期間を報告します。
- **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。
- 遅いディスクを除外し、ディスクが適度に速いことを確認するには、**etcd_disk_wal_fsync_duration_seconds_bucket** の 99 パーセンタイルが 10 ミリ秒未満であることを確認します。

OpenShift Container Platform クラスターを作成する前または後に etcd のハードウェアを検証するには、fio と呼ばれる I/O ベンチマークツールを使用できます。

前提条件

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

手順

- fio を実行し、結果を分析します。

- Podman を使用する場合は、次のコマンドを実行します。

```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/openshift-scale/etcd-perf
```

- Docker を使用する場合は、次のコマンドを実行します。

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

この出力では、実行からキャプチャーされた fsync メトリクスの 99 パーセンタイルの比較でディスクが 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。

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。

2.6. ETCD データのデフラグ

etcd 履歴の圧縮および他のイベントによりディスクの断片化が生じた後にディスク領域を回収するために、手動によるデフラグを定期的に実行する必要があります。

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。

etcd はデータをディスクに書き込むため、そのパフォーマンスはディスクのパフォーマンスに大きく依存します。etcd のデフラグは、毎月、月に 2 回、またはクラスターでの必要に応じて行うことを検討してください。**etcd_db_total_size_in_bytes** メトリクスをモニターして、デフラグが必要であるかどうかを判別することもできます。

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 の一覧を取得します。

   ```
   $ oc get pods -n openshift-etcd -o wide | grep -v quorum-guard | grep etcd
   ```

   出力例

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

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

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

   出力例

   ```
   Defaulting container name to etcdctl.
   Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
   all of the containers in this pod.
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+--------------------+---------+
   |         ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
   RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+--------------------+---------+
   |  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.4.9 |  104 MB |     false |      false |
   7 |      91624 |              91624 |        |
   | https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |  104 MB |     false |      false |
   7 |      91624 |              91624 |        |
   | https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  104 MB |      true |      false |
   7 |      91624 |              91624 |        |
   +---------------------------+------------------+---------+---------+-----------+------------+-----------
   +------------+--------------------+---------+
   ```

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

2. etcd メンバーのデフラグ。

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

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

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

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

c. etcd メンバーのデフラグを実行します。

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

出力例

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

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

d. データベースサイズが縮小されていることを確認します。

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

出力例

```
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
|         ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |   3.4.9 |  104 MB |     false |      false |
7 |      91624 |              91624 |        |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |   3.4.9 |   41 MB |     false |      false |
7 |      91624 |              91624 |        | 1
| https://10.0.199.170:2379 | 9ac311f93915cc79 |   3.4.9 |  104 MB |      true |      false |
7 |      91624 |              91624 |        |
+---------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+--------------------+--------+
```

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

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

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

a. **NOSPACE** アラームがあるかどうかを確認します。

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

出力例

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

b. アラームをクリアします。

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

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

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

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

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

関連情報

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

2.8. モニタリングソリューションの移動

デフォルトでは、Prometheus、Grafana、および AlertManager が含まれる Prometheus Cluster Monitoring スタックはクラスターモニターリングをデプロイするためにデプロイされます。これは Cluster Monitoring Operator によって管理されます。このコンポーネント異なるマシンに移行するに

は、カスタム設定マップを作成し、これを適用します。

手順

1. 以下の **ConfigMap** 定義を **cluster-monitoring-configmap.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    grafana:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    kubeStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    telemeterClient:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    openshiftStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    thanosQuerier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
```

この設定マップを実行すると、モニタリングスタックのコンポーネントがインフラストラクチャーノードに再デプロイされます。

2. 新規の設定マップを適用します。

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

3. モニタリング Pod が新規マシンに移行することを確認します。

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

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

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

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

2.9. デフォルトレジストリーの移行

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

前提条件

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

手順

1. **config/instance** オブジェクトを表示します。

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

出力例

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

2. **config/instance** オブジェクトを編集します。

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

3. オブジェクトの **spec** セクションを以下の YAML のように変更します。

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

4. レジストリー Pod がインフラストラクチャーノードに移動していることを確認します。

   a. 以下のコマンドを実行して、レジストリー Pod が置かれているノードを特定します。

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

   b. ノードに指定したラベルがあることを確認します。

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

   コマンド出力を確認し、**node-role.kubernetes.io/infra** が **LABELS** 一覧にあることを確認します。

2.10. ルーターの移動

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

前提条件

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

手順

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

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

   コマンド出力は以下のテキストのようになります。

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

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

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

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

以下に示すように、**infra** ラベルを参照する **nodeSelector** スタンザを **spec** セクションに追加します。

```
  spec:
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/infra: ""
```

3. ルーター Pod が **infra** ノードで実行されていることを確認します。
   a. ルーター Pod の一覧を表示し、実行中の Pod のノード名をメモします。

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

出力例

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

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

   b. 実行中の Pod のノードのステータスを表示します。

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

1 1 Pod の一覧より取得した **<node_name>** を指定します。

出力例

```
NAME                        STATUS  ROLES         AGE   VERSION
ip-10-0-217-226.ec2.internal  Ready   infra,worker  17h   v1.20.0
```

ロールの一覧に **infra** が含まれているため、Pod は正しいノードで実行されます。

2.11. インフラストラクチャーノードのサイジング

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

ワーカーノードの数	CPU コア数	メモリー (GB)
25	4	16
100	8	32
250	16	128
500	32	128

重要

これらのサイジングの推奨内容は、クラスター全体に多数のオブジェクトを作成するスケーリングのテストに基づいています。これらのテストでは、一部のクラスターの最大値に達しいます。OpenShift Container Platform 4.7 クラスターでノード数が 250 および 500 の場合、これらの最大値は、10000 namespace および 61000 Pod、10000 デプロイメント、181000 シークレット、400 設定マップなどになります。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。ディスクサイズは保持期間によっても変わります。これらの要素を考慮し、これらに応じてサイズを設定する必要があります。

これらのサイジングの推奨内容は、クラスターのインストール時にインストールされるインフラストラクチャーコンポーネント (Prometheus、ルーターおよびレジストリー) についてのみ適用されます。ロギングは Day 2 の操作で、これらの推奨事項には含まれていません。

注記

OpenShift Container Platform 4.7 では、デフォルトで CPU コア (500 ミリコア) の半分がシステムによって予約されます (OpenShift Container Platform 3.11 以前のバージョンと比較)。これは、上記のサイジングの推奨内容に影響します。

2.12. 関連情報

- OpenShift Container Platform クラスターの最大値

第3章 IBM Z および LINUXONE 環境に推奨されるホストプラクティス

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

注記

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

注記

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

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 PAGE を無効にする方法

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) プロファイルを使用して THP を無効にする方法を説明します。

3.2.1. Node Tuning Operator (NTO) プロファイルを使用した THP の無効化

手順

1. 以下の NTO 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、**thp-s390-tuned.yaml** のようになります。

```
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 on IBM Z 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: 35
    profile: openshift-thp-never-worker
```

2. NTO プロファイルを作成します。

```
$ oc create -f thp-s390-tuned.yaml
```

3. アクティブなプロファイルの一覧を確認します。

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

4. プロファイルを削除します。

```
$ oc delete -f thp-s390-tuned.yaml
```

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
always madvise [never]
```

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 設定では、ネットワーク設定は選択したハイパーバイザーによって異なります。ワークロードとアプリケーションに応じて、最適なものは通常、ユースケースとトラフィックパターンによって異なります。

設定によっては、以下のベストプラクティスを考慮してください。

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

関連情報

- OpenShift Container Platform on IBM Z - Performance Experiences, Hints and Tips

- OpenShift Container Platform on IBM Z Networking Performance
- ノードのアフィニティールールを使用したノード上での Pod 配置の制御

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

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

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

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

フルパックミニディスクを使用する z/VM ベースの OpenShift Container Platform セットアップの場合、すべてのノードで HyperPAV エイリアスをアクティベートして MCO プロファイルを利用できます。コントロールプレーンノードおよびコンピュートノードの 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 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. VirtIO ネットワークインターフェイスに複数のキューを使用

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

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

```
<interface type="direct">
   <source network="net01"/>
   <model type="virtio"/>
   <driver ... queues="2"/>
</interface>
```

複数のキューは、ネットワークインターフェイス用に強化されたパフォーマンスを提供するように設計

されていますが、メモリーおよび CPU リソースも使用します。ビジーなインターフェイス用の 2 つのキューの定義を開始します。次に、トラフィックが少ないインターフェイスの場合は 2 つのキューを、ビジーなインターフェイスの場合は 3 つ以上のキューを試してください。

3.6.2. 仮想ブロックデバイスの I/O スレッドの使用

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

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

I/O スレッド仕様のサンプル

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

1. I/O スレッドの数。
2. ディスクデバイスのドライバー要素。

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

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

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

3.6.3. 仮想 SCSI デバイスの回避

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

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

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

3.6.4. ディスクについてのゲストキャッシュの設定

ホストではなく、ゲストでキャッシュするようにディスクデバイスを設定します。

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

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

3.6.5. メモリーバルーンデバイスを除外します。

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

- アクティブなプロファイルの一覧を確認します。

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

3.6.6. ホストスケジューラーの CPU 移行アルゴリズムの調整

重要

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

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

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

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

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

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

```
kernel.sched_migration_cost_ns=60000
```

3.6.7. cpuset cgroup コントローラーの無効化

注記

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

手順

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

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

5. 新しい設定を有効にするには、libvirtd デーモンを再起動する必要があります。
   a. すべての仮想マシンを停止します。
   b. 以下のコマンドを実行します。

   ```
   # systemctl restart libvirtd
   ```

   c. 仮想マシンを再起動します。

この設定は、ホストの再起動後も維持されます。

3.6.8. アイドル状態の仮想 CPU のポーリング期間の調整

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

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

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

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

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

関連情報

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

第4章 クラスタースケーリングに関する推奨プラクティス

重要

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

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

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

4.1. クラスターのスケーリングに関する推奨プラクティス

クラスターをノード数のより高い値にスケールアップする場合:

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

注記

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

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

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

注記

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

4.2. マシンセットの変更

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

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

注記

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

前提条件

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

手順

1. マシンセットを編集します。

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

2. マシンセットを **0** にスケールダウンします。

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

または、以下を実行します。

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

マシンが削除されるまで待機します。

3. マシンセットを随時スケールアップします。

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

または、以下を実行します。

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

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

4.3. マシンのヘルスチェック

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

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

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

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

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

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

チェックを停止するには、リソースを削除します。

4.3.1. マシンヘルスチェックのデプロイ時の制限

マシンヘルスチェックをデプロイする前に考慮すべき制限事項があります。

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

4.4. サンプル MACHINEHEALTHCHECK リソース

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

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
```

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

① デプロイするマシンヘルスチェックの名前を指定します。

②③ チェックする必要のあるマシンプールのラベルを指定します。

④ 追跡するマシンセットを **<cluster_name>-<label>-<zone>** 形式で指定します。たとえば、**prod-node-us-east-1a** とします。

⑤⑥ ノードの状態のタイムアウト期間を指定します。タイムアウト期間の条件が満たされると、マシンは修正されます。タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。

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

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

注記

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

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

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

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

重要

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

適切な **maxUnhealthy** 値は、デプロイするクラスターの規模や、**MachineHealthCheck** が対応するマシンの数によって異なります。たとえば、**maxUnhealthy** 値を使用して複数のアベイラビリティーゾーン間で複数のマシンセットに対応でき、ゾーン全体が失われると、**maxUnhealthy** の設定によりクラスター内で追加の修復を防ぐことができます。

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

4.4.1.1. 絶対値を使用した maxUnhealthy の設定

maxUnhealthy が **2** に設定される場合:

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

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

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

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

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

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

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

注記

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

4.5. MACHINEHEALTHCHECK リソースの作成

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

前提条件

- **oc** コマンドラインインターフェイスをインストールします。

手順

1. マシンヘルスチェックの定義を含む **healthcheck.yml** ファイルを作成します。

2. **healthcheck.yml** ファイルをクラスターに適用します。

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

第5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。

5.1. NODE TUNING OPERATOR について

Node Tuning Operator は、Tuned デーモンのオーケストレーションによるノードレベルのチューニングの管理に役立ちます。ほとんどの高パフォーマンスアプリケーションでは、一定レベルのカーネルのチューニングが必要です。Node Tuning Operator は、ノードレベルの sysctl の統一された管理インターフェイスをユーザーに提供し、ユーザーが指定するカスタムチューニングを追加できるよう柔軟性を提供します。

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

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

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

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

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

手順

1. 以下を実行します。

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

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

警告

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

5.3. 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:
  - name: "openshift"
    data: |
      [main]
      summary=Optimize systems running OpenShift (parent profile)
      include=${f:virt_check:virtual-guest:throughput-performance}

      [selinux]
      avc_cache_threshold=8192

      [net]
      nf_conntrack_hashsize=131072

      [sysctl]
      net.ipv4.ip_forward=1
      kernel.pid_max=>4194304
      net.netfilter.nf_conntrack_max=1048576
      net.ipv4.conf.all.arp_announce=2
      net.ipv4.neigh.default.gc_thresh1=8192
      net.ipv4.neigh.default.gc_thresh2=32768
      net.ipv4.neigh.default.gc_thresh3=65536
      net.ipv6.neigh.default.gc_thresh1=8192
      net.ipv6.neigh.default.gc_thresh2=32768
      net.ipv6.neigh.default.gc_thresh3=65536
      vm.max_map_count=262144

      [sysfs]
      /sys/module/nvme_core/parameters/io_timeout=4294967295
      /sys/module/nvme_core/parameters/max_retries=10

  - name: "openshift-control-plane"
    data: |
```

```
      [main]
      summary=Optimize systems running OpenShift control plane
      include=openshift

      [sysctl]
      # ktune sysctl settings, maximizing i/o throughput
      #
      # Minimal preemption granularity for CPU-bound tasks:
      # (default: 1 msec#  (1 + ilog(ncpus)), units: nanoseconds)
      kernel.sched_min_granularity_ns=10000000
      # The total time the scheduler will consider a migrated process
      # "cache hot" and thus less likely to be re-migrated
      # (system default is 500000, i.e. 0.5 ms)
      kernel.sched_migration_cost_ns=5000000
      # SCHED_OTHER wake-up granularity.
      #
      # Preemption granularity when tasks wake up.  Lower the value to
      # improve wake-up latency and throughput for latency critical tasks.
      kernel.sched_wakeup_granularity_ns=4000000

  - name: "openshift-node"
    data: |
      [main]
      summary=Optimize systems running OpenShift nodes
      include=openshift

      [sysctl]
      net.ipv4.tcp_fastopen=3
      fs.inotify.max_user_watches=65536
      fs.inotify.max_user_instances=8192

  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
```

5.4. TUNED プロファイルが適用されていることの確認

この手順を使用して、すべてのノードに適用される Tuned プロファイルを確認します。

手順

1. 各ノードで実行されている Tuned Pod を確認します。

```
$ oc get pods -n openshift-cluster-node-tuning-operator -o wide
```

出力例

```
NAME                                            READY   STATUS    RESTARTS   AGE    IP            NODE
```

```
NOMINATED NODE   READINESS GATES
cluster-node-tuning-operator-599489d4f7-k4hw4   1/1     Running   0          6d2h   10.129.0.76
ip-10-0-145-113.eu-west-3.compute.internal   <none>           <none>
tuned-2jkzp                               1/1     Running   1          6d3h   10.0.145.113   ip-10-0-145-
113.eu-west-3.compute.internal   <none>           <none>
tuned-g9mkx                               1/1     Running   1          6d3h   10.0.147.108   ip-10-0-
147-108.eu-west-3.compute.internal   <none>           <none>
tuned-kbxsh                               1/1     Running   1          6d3h   10.0.132.143   ip-10-0-132-
143.eu-west-3.compute.internal   <none>           <none>
tuned-kn9x6                               1/1     Running   1          6d3h   10.0.163.177   ip-10-0-163-
177.eu-west-3.compute.internal   <none>           <none>
tuned-vvxwx                               1/1     Running   1          6d3h   10.0.131.87    ip-10-0-131-
87.eu-west-3.compute.internal    <none>           <none>
tuned-zqrwq                               1/1     Running   1          6d3h   10.0.161.51    ip-10-0-161-
51.eu-west-3.compute.internal    <none>           <none>
```

2. 各 Pod から適用されるプロファイルを抽出し、それらを直前の一覧に対して一致させます。

```
$ for p in `oc get pods -n openshift-cluster-node-tuning-operator -l openshift-app=tuned -
o=jsonpath='{range .items[*]}{.metadata.name} {end}'`; do printf "\n*** $p ***\n" ; oc logs
pod/$p -n openshift-cluster-node-tuning-operator | grep applied; done
```

出力例

```
*** tuned-2jkzp ***
2020-07-10 13:53:35,368 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied

*** tuned-g9mkx ***
2020-07-10 14:07:17,089 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:29,005 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied
2020-07-10 16:00:19,006 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 16:00:48,989 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-kbxsh ***
2020-07-10 13:53:30,565 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:30,199 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-kn9x6 ***
2020-07-10 14:10:57,123 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node' applied
2020-07-10 15:56:28,757 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-node-es' applied

*** tuned-vvxwx ***
2020-07-10 14:11:44,932 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied
```

```
*** tuned-zqrwq ***
2020-07-10 14:07:40,246 INFO     tuned.daemon.daemon: static tuning from profile
'openshift-control-plane' applied
```

5.5. 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一覧です。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一覧表示します。

```
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一覧です。

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

一覧の個別項目:

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

1. オプション:
2. キー/値の **MachineConfig** ラベルのディクショナリー。キーは一意である必要があります。
3. 省略する場合は、優先度の高いプロファイルが最初に一致するか、または **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. オプションの一覧。
5. プロファイルの順序付けの優先度。数値が小さいほど優先度が高くなります (**0** が最も高い優先度になります)。
6. 一致に適用する TuneD プロファイル。例: **tuned_profile_1**
7. オプションのオペランド設定。
8. TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は **true**、オフの場合は **false** です。デフォルトは **false** です。

<match> は、以下のように再帰的に定義されるオプションの一覧です。

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

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

4 オプションの **<match>** 一覧。

<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、**<match>** の一覧全体が **true** に評価されます。そのため、一覧は論理 OR 演算子として機能します。

machineConfigLabels が定義されている場合、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、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。

一覧項目の **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、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、チューニングされたオペランドが同じマシン設定プールを共有する 2 つ以上のノードの競合するカーネルパラメーターを計算する可能性があります。

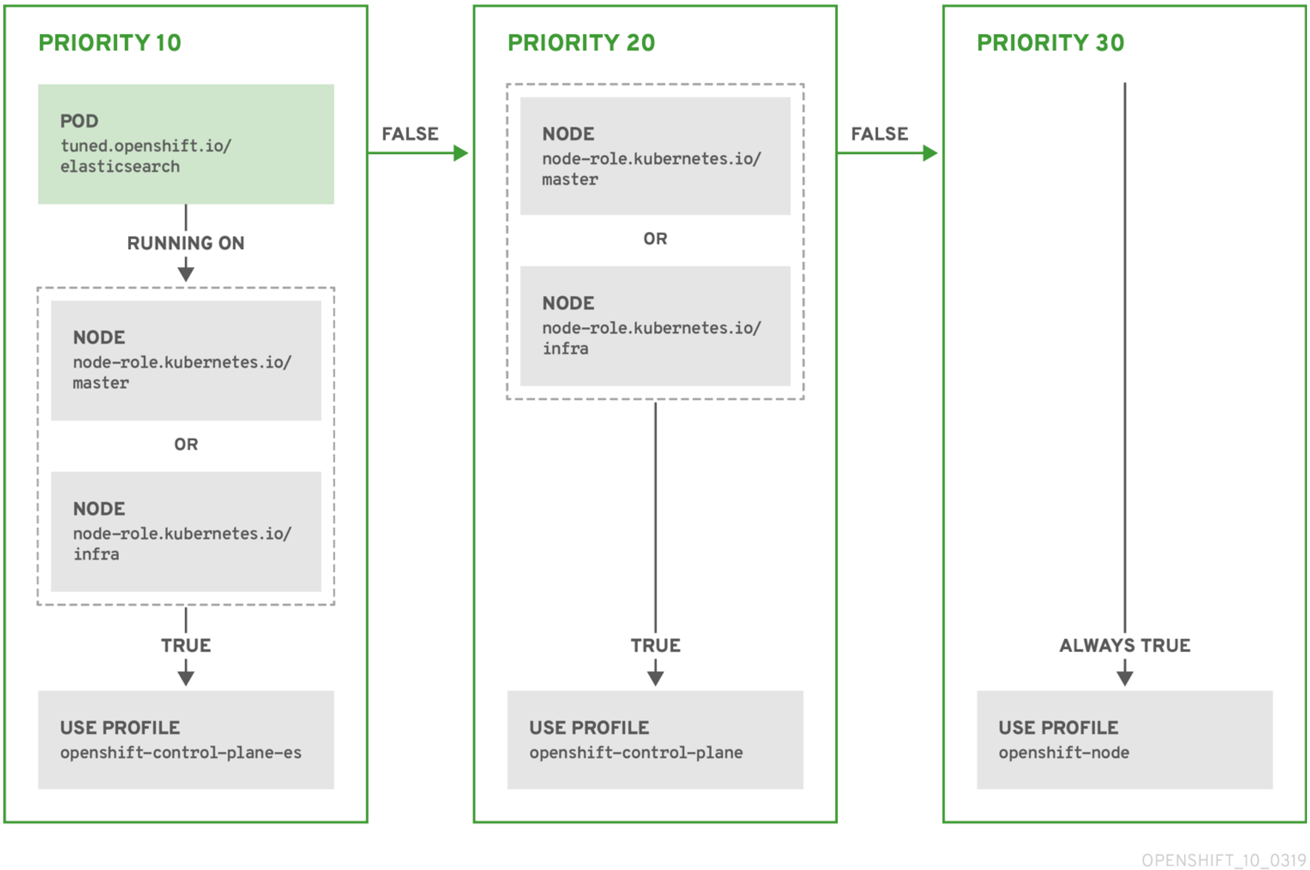
例: ノード/Pod ラベルベースのマッチング

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

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

優先順位が **10** のプロファイルのラベルが一致した場合、**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、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チューニング済み Pod が **node-role.kubernetes.io/master** または **node-role.kubernetes.io/infra** ラベルを持つノードで実行される場合に適用されます。

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



例: マシン設定プールベースのマッチング

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

  recommend:
```

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

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

5.6. カスタムチューニングの例

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。管理者として、以下のコマンドを使用してカスタムの Tune CR を作成します。

カスタムチューニングの例

```
$ oc create -f- <<_EOF_
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
_EOF_
```

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。

5.7. 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

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。

- bootloader
- script
- systemd

詳細は、利用可能な Tuned プラグイン および Tuned の使用 を参照してください。

第6章 クラスターローダーの使用

クラスターローダーとは、クラスターに対してさまざまなオブジェクトを多数デプロイするツールであり、ユーザー定義のクラスターオブジェクトを作成します。クラスターローダーをビルド、設定、実行して、さまざまなクラスターの状態にある OpenShift Container Platform デプロイメントのパフォーマンスメトリクスを測定します。

6.1. クラスターローダーのインストール

手順

1. コンテナーイメージをプルするには、以下を実行します。

```
$ podman pull quay.io/openshift/origin-tests:4.7
```

6.2. クラスターローダーの実行

前提条件

- リポジトリーは認証を要求するプロンプトを出します。レジストリーの認証情報を使用すると、一般的に利用できないイメージにアクセスできます。インストールからの既存の認証情報を使用します。

手順

1. 組み込まれているテスト設定を使用してクラスターローダーを実行し、5 つのテンプレートビルドをデプロイして、デプロイメントが完了するまで待ちます。

```
$ podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config:z -i \
quay.io/openshift/origin-tests:4.7 /bin/bash -c 'export KUBECONFIG=/root/.kube/config && \
openshift-tests run-test "[sig-scalability][Feature:Performance] Load cluster \
should populate the cluster [Slow][Serial] [Suite:openshift]"'
```

または、**VIPERCONFIG** の環境変数を設定して、ユーザー定義の設定でクラスターローダーを実行します。

```
$ podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config:z \
-v ${LOCAL_CONFIG_FILE_PATH}:/root/configs/:z \
-i quay.io/openshift/origin-tests:4.7 \
/bin/bash -c 'KUBECONFIG=/root/.kube/config VIPERCONFIG=/root/configs/test.yaml \
openshift-tests run-test "[sig-scalability][Feature:Performance] Load cluster \
should populate the cluster [Slow][Serial] [Suite:openshift]"'
```

この例では、 **${LOCAL_KUBECONFIG}** はローカルファイルシステムの **kubeconfig** のパスを参照します。さらに、**${LOCAL_CONFIG_FILE_PATH}** というディレクトリーがあり、これは **test.yaml** という設定ファイルが含まれるコンテナーにマウントされます。また、**test.yaml** が外部テンプレートファイルや podspec ファイルを参照する場合、これらもコンテナーにマウントされる必要があります。

6.3. クラスターローダーの設定

このツールは、複数のテンプレートや Pod を含む namespace (プロジェクト) を複数作成します。

6.3.1. クラスターローダー設定ファイルの例

クラスターローダーの設定ファイルは基本的な YAML ファイルです。

```
provider: local (1)
ClusterLoader:
  cleanup: true
  projects:
    - num: 1
      basename: clusterloader-cakephp-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: cakephp-mysql.json

    - num: 1
      basename: clusterloader-dancer-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: dancer-mysql.json

    - num: 1
      basename: clusterloader-django-postgresql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: django-postgresql.json

    - num: 1
      basename: clusterloader-nodejs-mongodb
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: quickstarts/nodejs-mongodb.json

    - num: 1
      basename: clusterloader-rails-postgresql
      tuning: default
      templates:
        - num: 1
          file: rails-postgresql.json

  tuningsets: (2)
    - name: default
      pods:
        stepping: (3)
          stepsize: 5
          pause: 0 s
        rate_limit: (4)
          delay: 0 ms
```

1. エンドツーエンドテストのオプション設定。**local** に設定して、過剰に長いログメッセージを回避します。
2. このチューニングセットでは、速度制限やステップ設定、複数の Pod バッチ作成、セット間での一時停止などが可能になります。クラスターローダーは、以前のステップが完了したことをモニターリングしてから、続行します。
3. ステップ設定では、オブジェクトが **N** 個作成されてから、**M** 秒間一時停止します。
4. 速度制限は、次のオブジェクトを作成するまで **M** ミリ秒間待機します。

この例では、外部テンプレートファイルや Pod 仕様ファイルへの参照もコンテナーにマウントされていることを前提とします。

重要

Microsoft Azure でクラスターローダーを実行している場合、**AZURE_AUTH_LOCATION** 変数を、インストーラーディレクトリーにある **terraform.azure.auto.tfvars.json** の出力が含まれるファイルに設定する必要があります。

6.3.2. 設定フィールド

表6.1 クラスターローダーの最上位のフィールド

フィールド	説明
cleanup	**true** または **false** に設定します。設定ごとに 1 つの定義を設定します。**true** に設定すると、**cleanup** は、テストの最後にクラスターローダーが作成した namespace (プロジェクト) すべてを削除します。
projects	1 つまたは多数の定義が指定されたサブオブジェクト。**projects** の下に、作成する各 namespace が定義され、**projects** には必須のサブヘッダーが複数指定されます。
tuningsets	設定ごとに 1 つの定義が指定されたサブオブジェクト。**tuningsets** では、チューニングセットを定義して、プロジェクトやオブジェクト作成に対して設定可能なタイミングを追加することができます (Pod、テンプレートなど)。
sync	設定ごとに 1 つの定義が指定されたオプションのサブオブジェクト。オブジェクト作成時に同期できるかどうかについて追加します。

表6.2 projects の下にあるフィールド

フィールド	説明
num	整数。作成するプロジェクト数の 1 つの定義。
basename	文字列。プロジェクトのベース名の定義。競合が発生しないように、同一の namespace の数が **Basename** に追加されます。
tuning	文字列。オブジェクトに適用するチューニングセットの 1 つの定義。 これは対象の namespace にデプロイします。
ifexists	**reuse** または **delete** のいずれかが含まれる文字列。ツールが実行時に作成するプロジェクトまたは namespace の名前と同じプロジェクトまたは namespace を見つける場合のツールの機能を定義します。
configmaps	キーと値のペア一覧。キーは設定マップの名前で、値はこの設定マップの作成元のファイルへのパスです。
secrets	キーと値のペア一覧。キーはシークレットの名前で、値はこのシークレットの作成元のファイルへのパスです。
pods	デプロイする Pod の 1 つまたは多数の定義を持つサブオブジェクト
templates	デプロイするテンプレートの 1 つまたは多数の定義を持つサブオブジェクト

表6.3 pods および templates のフィールド

フィールド	説明
num	整数。デプロイする Pod またはテンプレート数。
image	文字列。プルが可能なリポジトリーに対する Docker イメージの URL
basename	文字列。作成するテンプレート (または Pod) のベース名の 1 つの定義。
file	文字列。ローカルファイルへのパス。 作成する Pod 仕様またはテンプレートのいずれかです。

フィールド	説明
parameters	キーと値のペア。**parameters** の下で、Pod またはテンプレートでオーバーライドする値の一覧を指定できます。

表6.4 tuningsets の下にあるフィールド

フィールド	説明
name	文字列。チューニングセットの名前。 プロジェクトのチューニングを定義する時に指定した名前と一致します。
pods	Pod に適用される **tuningsets** を特定するサブオブジェクト
templates	テンプレートに適用される **tuningsets** を特定するサブオブジェクト

表6.5 tuningsets pods または tuningsets templates の下にあるフィールド

フィールド	説明
stepping	サブオブジェクト。ステップ作成パターンでオブジェクトを作成する場合に使用するステップ設定。
rate_limit	サブオブジェクト。オブジェクト作成速度を制限するための速度制限チューニングセットの設定。

表6.6 tuningsets pods または tuningsets templates、stepping の下にあるフィールド

フィールド	説明
stepsize	整数。オブジェクト作成を一時停止するまでに作成するオブジェクト数。
pause	整数。**stepsize** で定義したオブジェクト数を作成後に一時停止する秒数。
timeout	整数。オブジェクト作成に成功しなかった場合に失敗するまで待機する秒数。
delay	整数。次の作成要求まで待機する時間 (ミリ秒)。

表6.7 sync の下にあるフィールド

フィールド	説明
server	**enabled** および **port** フィールドを持つサブオブジェクト。ブール値 **enabled** を指定すると、Pod を同期するために HTTP サーバーを起動するかどうか定義します。**port** の整数はリッスンする HTTP サーバーポートを定義します (デフォルトでは **9090**)。
running	ブール値。**selectors** に一致するラベルが指定された Pod が **Running** の状態になるまで待機します。
succeeded	ブール値。**selectors** に一致するラベルが指定された Pod が **Completed** の状態になるまで待機します。
selectors	**Running** または **Completed** の状態の Pod に一致するセレクター一覧
timeout	文字列。**Running** または **Completed** の状態の Pod を待機してから同期をタイムアウトするまでの時間。**0** 以外の値は、単位 [ns|us|ms|s|m|h] を使用してください。

6.4. 既知の問題

- クラスターローダーは設定なしで呼び出される場合に失敗します。(BZ#1761925)
- **IDENTIFIER** パラメーターがユーザーテンプレートで定義されていない場合には、テンプレートの作成は **error: unknown parameter name "IDENTIFIER"** エラーを出して失敗します。テンプレートをデプロイする場合は、このエラーが発生しないように、以下のパラメーターをテンプレートに追加してください。

```
{
  "name": "IDENTIFIER",
  "description": "Number to append to the name of resources",
  "value": "1"
}
```

  Pod をデプロイする場合は、このパラメーターを追加する必要はありません。

第7章 CPU マネージャーの使用

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

7.1. 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。
- **static** このポリシーにより、特定のリソース特性を持つ Pod の CPU アフィニティーを増やし、これらの Pod のノードにおける排他性を付与することができます。

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 2 これらの設定は、**KubeletConfig** CR を作成する際に定義されたものです。

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

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

出力例

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

9. Pod を作成します。

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

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

```
# oc describe pod cpumanager
```

出力例

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

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

```
# ├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
└─kubepods.slice
  ├─kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
  │ ├─crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
  │ └─32706 /pause
```

QoS (quality of service) 階層 **Guaranteed** の Pod は、**kubepods.slice** に配置されます。他の QoS 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 一覧を確認します。

```
# 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 階層にある 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
```

第8章 TOPOLOGY MANAGER の使用

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、または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。

注記

Topology Manager を使用するには、**static** ポリシーで CPU マネージャーを使用する必要があります。CPU マネージャーの詳細は、CPU マネージャーの使用 を参照してください。

8.1. TOPOLOGY MANAGER ポリシー

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

注記

CPU リソースを **Pod** 仕様の他の要求されたリソースと調整するには、CPU マネージャーを **static** CPU マネージャーポリシーで有効にする必要があります。

Topology Manager は、**cpumanager-enabled** 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。

8.2. TOPOLOGY MANAGER のセットアップ

Topology Manager を使用するには、 **cpumanager-enabled** 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。スケーラビリティーおよびパフォーマンスセクションの CPU マネージャーの使用を参照してください。

手順

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

1. **cpumanager-enabled** カスタムリソース (CR) 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 **static** である必要があります。

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。

関連情報

- CPU マネージャーの詳細は、CPU マネージャーの使用 を参照してください。

8.3. 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、利用可能な CPU のトポロジーを返す CPU マネージャーの静的ポリシーを確認します。また Topology Manager はデバイスマネージャーを確認し、example.com/device の利用可能なデバイスのトポロジーを検出します。

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9章 CLUSTER MONITORING OPERATOR のスケーリング

OpenShift Container Platform は、Cluster Monitoring Operator が収集し、Prometheus ベースのモニター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、Grafana という 1 つのダッシュボードインターフェイスでシステムリソース、コンテナーおよびコンポーネントのメトリクスを表示できます。

重要

Prometheus の割り当てられた PVC を使用してクラスターモニターリングを実行している場合、クラスターのアップグレード時に OOM による強制終了が生じる可能性があります。永続ストレージが Prometheus 用に使用される場合、Prometheus のメモリー使用量はクラスターのアップグレード時、およびアップグレードの完了後の数時間で 2 倍になります。OOM による強制終了の問題を回避するには、ワーカーノードで、アップグレード前に利用可能なメモリーのサイズを 2 倍にできるようにします。たとえば、推奨される最小ノード (RAM が 8 GB の 2 コア) でモニターリングを実行している場合は、メモリーを 16 GB に増やします。詳細は、BZ#1925061 を参照してください。

9.1. PROMETHEUS データベースのストレージ要件

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

注記

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

表9.1 クラスター内のノード/Pod の数に基づく Prometheus データベースのストレージ要件

ノード数	Pod 数	1 日あたりの Prometheus ストレージの増加量	15 日ごとの Prometheus ストレージの増加量	RAM 領域 (スケールサイズに基づく)	ネットワーク (tsdb チャンクに基づく)
50	1800	6.3 GB	94 GB	6 GB	16 MB
100	3600	13 GB	195 GB	10 GB	26 MB
150	5400	19 GB	283 GB	12 GB	36 MB
200	7200	25 GB	375 GB	14 GB	46 MB

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

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

注記

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

OpenShift Container Platform についての推奨事項

- 3 つ以上のインフラストラクチャー (infra) ノードを使用します。
- NVMe (non-volatile memory express) ドライブを搭載した 3 つ以上の **openshift-container-storage** ノードを使用します。

9.2. クラスターモニターリングの設定

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

手順

Prometheus のストレージ容量を拡張するには、以下を実行します。

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

```
apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} 1
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: {{STORAGE_CLASS}} 2
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} 3
    alertmanagerMain:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: {{STORAGE_CLASS}} 4
          resources:
            requests:
              storage: {{ALERTMANAGER_STORAGE_SIZE}} 5
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
```

1 標準の値は **PROMETHEUS_RETENTION_PERIOD=15d** になります。時間は、接尾辞 s、m、h、d のいずれかを使用する単位で測定されます。

2 4 クラスターのストレージクラス。

3 標準の値は **PROMETHEUS_STORAGE_SIZE=2000Gi** です。ストレージの値には、接尾辞 E、P、T、G、M、K のいずれかを使用した単純な整数または固定小数点整数を使用で

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

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

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

第10章 オブジェクトの最大値に合わせた環境計画

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

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

重要

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

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

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

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

最大値のタイプ	3.x テスト済みの最大値	4.x テスト済みの最大値
ノード数	2,000	2,000
Pod 数 [1]	150,000	150,000
ノードあたりの Pod 数	250	500 [2]
コアあたりの Pod 数	デフォルト値はありません。	デフォルト値はありません。
namespace 数 [3]	10,000	10,000
ビルド数	10,000(デフォルト Pod RAM 512 Mi)- Pipeline ストラテジー	10,000(デフォルト Pod RAM 512 Mi)- Source-to-Image (S2I) ビルドストラテジー
namespace ごとの Pod 数 [4]	25,000	25,000
Ingress Controller ごとのルートとバックエンドの数	ルーターあたり 2,000	ルーターあたり 2,000
シークレットの数	80,000	80,000
config map の数	90,000	90,000

最大値のタイプ	3.x テスト済みの最大値	4.x テスト済みの最大値
サービス数 [5]	10,000	10,000
namespace ごとのサービス数	5,000	5,000
サービスごとのバックエンド数	5,000	5,000
namespace ごとのデプロイメント数 [4]	2,000	2,000
ビルド設定の数	12,000	12,000
シークレットの数	40,000	40,000
カスタムリソース定義 (CRD) の数	デフォルト値はありません。	512 [6]

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

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

3. 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お勧めします。

4. システムには、状態の変更に対する対応として特定の namespace にある全オブジェクトに対して反復する多数のコントロールループがあります。単一の namespace に特定タイプのオブジェクトの数が多くなると、ループのコストが上昇し、特定の状態変更を処理する速度が低下します。この制限については、アプリケーションの各種要件を満たすのに十分な CPU、メモリー、およびディスクがシステムにあることが前提となっています。

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

6. OpenShift Container Platform には、OpenShift Container Platform によってインストールされたもの、OpenShift Container Platform と統合された製品、およびユーザー作成の CRD を含め、合計 512 のカスタムリソース定義 (CRD) の制限があります。512 を超える CRD が作成されている場合は、**oc** コマンドリクエストのスロットリングが適用される可能性があります。

注記

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

10.2. 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

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
マスター/etcd [1]	r5.4xlarge	16	128	io1	220 / 3000	3	us-west-2
インフラ [2]	m5.12xlarge	48	192	gp2	100	3	us-west-2
ワークロード [3]	m5.4xlarge	16	64	gp2	500 [4]	1	us-west-2
ワーカー	m5.2xlarge	8	32	gp2	100	3/25/250/500 [5]	us-west-2

1. 3000 IOPS を持つ io1 ディスクは、etcd が I/O 集約型であり、かつレイテンシーの影響を受けやすいため、マスター/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、モニターリング、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。

10.3. テスト済みのクラスターの最大値に基づく環境計画

重要

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

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

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

環境の計画時に、ノードに配置できる Pod 数を判別します。

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

ノードあたりの現在の Pod の最大数は 250 です。ただし、ノードに適合する Pod 数はアプリケーション自体によって異なります。**アプリケーション要件に合わせて環境計画を立てる方法**で説明されているように、アプリケーションのメモリー、CPU およびストレージの要件を検討してください。

シナリオ例

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

```
2200 / 500 = 4.4
```

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

```
2200 / 20 = 110
```

ここで、

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

10.4. アプリケーション要件に合わせて環境計画を立てる方法

アプリケーション環境の例を考えてみましょう。

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

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

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

ノードのタイプ	数量	CPU	RAM (GB)
ノード (オプション 1)	100	4	16
ノード (オプション 2)	50	8	32
ノード (オプション 3)	25	16	64

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

アプリケーション Pod は環境変数または DNS のいずれかを使用してサービスにアクセスできます。環境変数を使用する場合、それぞれのアクティブなサービスについて、変数が Pod がノードで実行される際に kubelet によって挿入されます。クラスター対応の DNS サーバーは、Kubernetes API で新規サービスの有無を監視し、それぞれに DNS レコードのセットを作成します。DNS がクラスター全体で有効にされている場合、すべての Pod は DNS 名でサービスを自動的に解決できるはずです。DNS を使用したサービス検出は、5000 サービスを超える使用できる場合があります。サービス検出に環境変数を使用する場合、引数の一覧は 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
    portalIP: ''
    type: ClusterIP
    sessionAffinity: None
  status:
    loadBalancer: {}
parameters:
- name: IDENTIFIER
  description: Number to append to the name of resources
  value: '1'
  required: true
- name: IMAGE
  description: Image to use for deploymentConfig
  value: gcr.io/google-containers/pause-amd64:3.0
  required: false
```

```
- name: ENV_VALUE
  description: Value to use for environment variables
  generate: expression
  from: "[A-Za-z0-9]{255}"
  required: false
labels:
  template: deployment-config-template
```

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

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

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

第11章 ストレージの最適化

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

11.1. 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11.1 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 Container Platform レジストリーで使用するために設定できます。 • 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 プラグインを使用する場合に動的 PV のプロビジョニングをサポートします。

重要

現時点で、CNS は OpenShift Container Platform 4.7 ではサポートされていません。

11.2. 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11.2 設定可能な推奨ストレージ技術

ストレージタイプ	ROX1	RWX2	レジストリー	スケーリングされたレジストリー	メトリクス3	ロギング	アプリ
ブロック	はい[4]	いいえ	設定可能	設定不可	推奨	推奨	推奨
ファイル	はい[4]	○	設定可能	設定可能	設定可能[5]	設定可能[6]	推奨
オブジェクト	○	○	推奨	推奨	設定不可	設定不可	設定不可[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

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

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

注記

スケーリングされたレジストリーとは、2 つ以上の Pod レプリカが稼働する OpenShift Container Platform レジストリーのことです。

11.2.1. 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、メトリクスストレージの 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。

11.2.1.1. レジストリー

スケーリングなし/高可用性 (HA) ではない OpenShift Container Platform レジストリークラスターのデプロイメント:

- 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、実稼働環境のワークロードを処理する OpenShift Container Platform 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。

11.2.1.2. スケーリングされたレジストリー

スケーリングされた/高可用性 (HA) の OpenShift Container Platform レジストリーのクラスターデプロイメント:

- 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。
- Amazon Simple Storage Service (Amazon S3)、Google Cloud Storage (GCS)、Microsoft Azure Blob Storage、および OpenStack Swift はサポートされます。
- 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。

11.2.1.3. メトリクス

OpenShift Container Platform がホストするメトリクス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クス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。

11.2.1.4. ロギング

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

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

11.2.1.5. 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。

11.2.2. 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お勧めしません。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。

11.3. 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11.3 OpenShift Container Platform データを保存するメインディレクトリー

ディレクトリー	注記	サイジング	予想される拡張
/var/lib/etcd	データベースを保存する際に etcd ストレージに使用されます。	20 GB 未満。 データベースは、最大 8 GB まで拡張できます。	環境と共に徐々に拡張します。メタデータのみを格納します。 メモリーに 8 GB が追加されるたびに 20-25 GB を追加します。
/var/lib/containers	これは CRI-O ランタイムのマウントポイントです。アクティブなコンテナーランタイム (Pod を含む) およびローカルイメージのストレージに使用されるストレージです。レジストリーストレージには使用されません。	16 GB メモリーの場合、1 ノードにつき 50 GB。このサイジングは、クラスターの最小要件の決定には使用しないでください。 メモリーに 8 GB が追加されるたびに 20-25 GB を追加します。	拡張は実行中のコンテナーの容量によって制限されます。
/var/lib/kubelet	Pod の一時ボリュームストレージです。これには、ランタイムにコンテナーにマウントされる外部のすべての内容が含まれます。環境変数、kube シークレット、および永続ボリュームでサポートされていないデータボリュームが含まれます。	変動あり。	ストレージを必要とする Pod が永続ボリュームを使用している場合は最小になります。一時ストレージを使用する場合はすぐに拡張する可能性があります。
/var/log	**すべてのコンポーネントのログファイルです。**	**10 から 30 GB。**	**ログファイルはすぐに拡張する可能性があります。サイズは拡張するディスク別に管理するか、ログローテーションを使用して管理できます。**

第12章 ルーティングの最適化

OpenShift Container Platform HAProxy ルーターは、パフォーマンスを最適化するためにスケーリングします。

12.1. 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 コントローラーまたはルーターは、宛先が OpenShift Container Platform サービスのすべての外部トラフィックに対する Ingress ポイントです。

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

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

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

HTTP keep-alive モードのシナリオの場合:

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

HTTP close (keep-alive なし) のシナリオの場合:

暗号化	LoadBalancerService	HostNetwork
なし	5719	8273
edge	2729	4069

暗号化	LoadBalancerService	HostNetwork
passthrough	4121	5344
re-encrypt	2320	2941

ROUTER_THREADS=4 が設定されたデフォルトの Ingress コントローラー設定が使用され、2 つの異なるエンドポイントの公開ストラテジー (LoadBalancerService/HostNetwork) がテストされています。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 の場合は、単一の HAProxy ルーターがページサイズが 8kB でも、1 Gbit 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層により発生し、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、使用されるて技術に応じて 5 から 1000 のアプリケーションのルーターをサポートします。Ingress コントローラーのパフォーマンスは、言語や静的コンテンツと動的コンテンツの違いを含め、その背後にあるアプリケーションの機能およびパフォーマンスによって制限される可能性があります。

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

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

12.2. INGRESS コントローラー (ルーター) のパフォーマンスの最適化

OpenShift Container Platform では、環境変数 (**ROUTER_THREADS**、**ROUTER_DEFAULT_TUNNEL_TIMEOUT**、**ROUTER_DEFAULT_CLIENT_TIMEOUT**、**ROUTER_DEFAULT_SERVER_TIMEOUT**、および **RELOAD_INTERVAL**) を設定して Ingress コントローラーのデプロイメントを変更することをサポートしていません。

Ingress コントローラーのデプロイメントは変更できますが、Ingress Operator が有効にされている場合、設定は上書きされます。

第13章 ネットワークの最適化

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

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

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

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

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

1 Gbps 以上にするには、以下を実行してください。

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

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

13.1. 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。

SDN オーバーレイの MTU は、最低でも NIC MTU より 50 バイト少なくなければなりません。これは、SDN オーバーレイのヘッダーに相当します。そのため、通常のイーサネットネットワークでは、この値を **1450** に設定します。ジャンボフレームのイーサネットネットワークの場合は、これを **8950** に設定します。

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**
>
> 50 バイトのオーバーレイヘッダーは OpenShift SDN 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。

13.2. 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
  machineCIDR: 10.0.0.0/16
  networkType: OpenShiftSDN
  serviceNetwork:
  - 172.30.0.0/16
```

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

13.3. IPSEC の影響

ノードホストの暗号化、復号化に CPU 機能が使用されるので、使用する IP セキュリティーシステムにかかわらず、ノードのスループットおよび CPU 使用率の両方でのパフォーマンスに影響があります。

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

関連情報

- 高度なネットワーク設定パラメーターの変更
- OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーター
- OpenShift SDN デフォルト CNI ネットワークプロバイダーの設定パラメーター

第14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。

14.1. ベアメタルホストおよびノードについて

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

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

14.2. ベアメタルホストのメンテナンス

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

ベアメタルホストをメンテナンスモードに移行できます。ホストをメンテナンスモードに移行すると、スケジューラーはすべての管理ワークロードを対応するベアメタルノードから移動します。新しいワークロードは、メンテナンスモードの間はスケジュールされません。

Web コンソールでベアメタルホストのプロビジョニングを解除することができます。ホストのプロビジョニング解除により以下のアクションが実行されます。

1. ベアメタルホスト CR に **cluster.k8s.io/delete-machine: true** のアノテーションを付けます。
2. 関連するマシンセットをスケールダウンします。

> **注記**
>
> デーモンセットおよび管理対象外の静的 Pod を別のノードに最初に移動することなく、ホストの電源をオフにすると、サービスの中断やデータの損失が生じる場合があります。

関連情報

- コンピュートマシンのベアメタルへの追加

14.2.1. 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. 必要に応じて、ホストの電源管理を有効にします。これにより、OpenShift Container Platform はホストの電源状態を制御できます。
7. 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。
8. 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。
9. 利用可能なベアメタルホストの数に一致するようにレプリカ数をスケールアップします。**Compute → MachineSets** に移動し、**Actions** ドロップダウンメニューから **Edit Machine count** を選択してクラスター内のマシンレプリカ数を増やします。

注記

oc scale コマンドおよび適切なベアメタルマシンセットを使用して、ベアメタルノードの数を管理することもできます。

14.2.2. 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
  bootMACAddress: <host_boot_mac_address>
  hardwareProfile: unknown
```

(1) **credentialsName** は有効な **Secret** CR を参照する必要があります。**baremetal-operator** は、**credentialsName** で参照される有効な **Secret** なしに、ベアメタルホストを管理できません。シークレットの詳細および作成方法については、シークレットについて を参照してください。

4. **Create** を選択して YAML を保存し、新規ベアメタルホストを作成します。
5. 利用可能なベアメタルホストの数に一致するようにレプリカ数をスケールアップします。**Compute → MachineSets** に移動し、**Actions** ドロップダウンメニューから **Edit Machine count** を選択してクラスター内のマシン数を増やします。

注記

oc scale コマンドおよび適切なベアメタルマシンセットを使用して、ベアメタルノードの数を管理することもできます。

14.2.3. 利用可能なベアメタルホストの数へのマシンの自動スケーリング

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

前提条件

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

手順

1. **metal3.io/autoscale-to-hosts** アノテーションを追加して、自動スケーリング用に設定するマシンセットにアノテーションを付けます。**<machineset>** を、マシンセット名に置き換えます。

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

新しいスケーリングされたマシンが起動するまで待ちます。

注記

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

関連情報

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

第15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

15.1. 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。

15.2. 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。

関連情報

- Transparent Huge Page の設定

15.3. HUGE PAGE の設定

ノードは、OpenShift Container Platform クラスターで使用される Huge Page を事前に割り当てる必要があります。Huge Page を予約する方法は、ブート時とランタイム時に実行する 2 つの方法があります。ブート時の予約は、メモリーが大幅に断片化されていないために成功する可能性が高くなります。Node Tuning Operator は、現時点で特定のノードでの Huge Page のブート時の割り当てをサポートします。

15.3.1. ブート時

手順

ノードの再起動を最小限にするには、以下の手順の順序に従う必要があります。

1. ラベルを使用して同じ Huge Page 設定を必要とするすべてのノードにラベルを付けます。

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. 以下の内容でファイルを作成し、これに **hugepages-tuned-boottime.yaml** という名前を付けます。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 3
    name: openshift-node-hugepages

  recommend:
  - machineConfigLabels: 4
      machineconfiguration.openshift.io/role: "worker-hp"
    priority: 30
    profile: openshift-node-hugepages
```

1 チューニングされたリソースの **name** を **hugepages** に設定します。

2 Huge Page を割り当てる **profile** セクションを設定します。

3 一部のプラットフォームではさまざまなサイズの Huge Page をサポートするため、パラメーターの順序に注意してください。

4 マシン設定プールベースのマッチングを有効にします。

3. チューニングされた **hugepages** プロファイルの作成

```
$ 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
```

> **警告**
>
> この機能は、現在 Red Hat Enterprise Linux CoreOS (RHCOS) 8.x ワーカーノードでのみサポートされています。Red Hat Enterprise Linux (RHEL) 7.x ワーカーノードでは、チューニングされた **[bootloader]** プラグインは現時点でサポートされていません。

第16章 低レイテンシーのノード向けの PERFORMANCE ADDON OPERATOR

16.1. 低レイテンシー

Telco / 5G の領域でのエッジコンピューティングの台頭は、レイテンシーと輻輳を軽減し、アプリケーションのパフォーマンスを向上させる上で重要なロールを果たします。

簡単に言うと、レイテンシーは、データ (パケット) が送信側から受信側に移動し、受信側の処理後に送信側に戻るスピードを決定します。レイテンシーによる遅延を最小限に抑えた状態でネットワークアーキテクチャーを維持することが 5 G のネットワークパフォーマンス要件を満たすのに鍵となります。4G テクノロジーと比較し、平均レイテンシーが 50ms の 5G では、レイテンシーの数値を 1ms 以下にするようにターゲットが設定されます。このレイテンシーの減少により、ワイヤレスのスループットが 10 倍向上します。

Telco 領域にデプロイされるアプリケーションの多くは、ゼロパケットロスに耐えられる低レイテンシーを必要とします。パケットロスをゼロに調整すると、ネットワークのパフォーマンス低下させる固有の問題を軽減することができます。詳細は、Tuning for Zero Packet Loss in Red Hat OpenStack Platform (RHOSP) を参照してください。

エッジコンピューティングの取り組みは、レイテンシーの削減にも役立ちます。コンピュート能力が文字通りクラウドのエッジ上にあり、ユーザーの近く置かれること考えてください。これにより、ユーザーと離れた場所にあるデータセンター間の距離が大幅に削減されるため、アプリケーションの応答時間とパフォーマンスのレイテンシーが短縮されます。

管理者は、すべてのデプロイメントを可能な限り低い管理コストで実行できるように、多数のエッジサイトおよびローカルサービスを一元管理できるようにする必要があります。また、リアルタイムの低レイテンシーおよび高パフォーマンスを実現するために、クラスターの特定のノードをデプロイし、設定するための簡単な方法も必要になります。低レイテンシーノードは、Cloud-native Network Functions (CNF) や Data Plane Development Kit (DPDK) などのアプリケーションに役立ちます。

現時点で、OpenShift Container Platform はリアルタイムの実行および低レイテンシーを実現するために OpenShift Container Platform クラスターでソフトウェアを調整するメカニズムを提供します (約 20 マイクロ秒未満の応答時間)。これには、カーネルおよび OpenShift Container Platform の設定値のチューニング、カーネルのインストール、およびマシンの再設定が含まれます。ただし、この方法では 4 つの異なる Operator を設定し、手動で実行する場合に複雑であり、間違いが生じる可能性がある多くの設定を行う必要があります。

OpenShift Container Platform は、OpenShift アプリケーションの低レイテンシーパフォーマンスを実現するために自動チューニングを実装する Performance Addon Operator を提供します。クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更をより容易に実行することができます。管理者は、カーネルを kernel-rt、ハウスキーピング用に予約される CPU、ワークロードの実行に使用される CPU に更新するかどうかを指定できます。

16.2. PERFORMANCE ADDON OPERATOR のインストール

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

16.2.1. CLI を使用した Operator のインストール

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

前提条件

- ベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) 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。

手順

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-performance-addon-operator
```

   b. 以下のコマンドを実行して namespace を作成します。

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

2. 以下のオブジェクトを作成して、直前の手順で作成した namespace に Performance Addon Operator をインストールします。
   a. 以下の **OperatorGroup** CR を作成し、YAML を **pao-operatorgroup.yaml** ファイルに保存します。

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

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

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

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

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

出力例

```
4.7
```

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

Subscription の例

```
apiVersion: operators.coreos.com/v1alpha1
```

```
kind: Subscription
metadata:
  name: openshift-performance-addon-operator-subscription
  namespace: openshift-performance-addon-operator
spec:
  channel: "<channel>" 1
  name: performance-addon-operator
  source: redhat-operators 2
  sourceNamespace: openshift-marketplace
```

1. **.status.defaultChannel** パラメーターの直前の手順で取得した値を指定します。
2. **redhat-operators** 値を指定する必要があります。

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

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

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

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

16.2.2. Web コンソールを使用した Performance Addon Operator のインストール

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

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

手順

1. OpenShift Container Platform Web コンソールを使用して Performance Addon Operator をインストールします。
    a. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
    b. 利用可能な Operator の一覧から **Performance Addon Operator** を選択してから **Install** をクリックします。
    c. **Install Operator** ページで、**All namespaces on the cluster**を選択します。次に、**Install** をクリックします。
2. オプション: performance-addon-operator が正常にインストールされていることを確認します。
    a. **Operators** → **Installed Operators** ページに切り替えます。
    b. **Performance Addon Operator** が **openshift-operators** プロジェクトに **Succeeded** の **Status** でリストされていることを確認します。

> **注記**
>
> インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが成功し、**Succeeded** メッセージが表示された場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合は、さらにトラブルシューティングを行うことができます。

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

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

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

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

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

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

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

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

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

前提条件

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

手順

1. OpenShift Web コンソールにアクセスし、 **Operators → Installed Operators** に移動します。

2. **Performance Addon Operator** をクリックし、**Operator Details** ページを開きます。

3. **Subscription** タブをクリックし、**Subscription Overview** ページを開きます。

4. **Channel** ペインで、バージョン番号の右側にある鉛筆アイコンをクリックし、**Change Subscription Update Channel** ウィンドウを開きます。

5. 次のマイナーバージョンを選択します。たとえば、Performance Addon Operator 4.7 にアップグレードする場合は、**4.7** を選択します。

6. **Save** をクリックします。

7. **Operators → Installed Operators** に移動してアップグレードのステータスを確認します。以下の **oc** コマンドを実行してステータスを確認することもできます。

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

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

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

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift クラスターにログインします。

手順

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

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

2. Operator Lifecycle Manager (OLM) が変更を処理するまで待機します。

3. OperatorGroup CR の変更が正常に適用されていることを確認します。**OperatorGroup** CR の **spec** 要素が削除されていることを確認します。

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

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

16.3.2. アップグレードステータスの監視

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

> **注記**
>
> **PHASE** および状態の値は利用可能な情報に基づく近似値になります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) をインストールしている。

手順

1. 以下のコマンドを実行します。

```
$ oc get csv
```

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

```
VERSION    REPLACES                                  PHASE
4.7.0      performance-addon-operator.v4.6.0              Installing
4.6.0                                          Replacing
```

3. **get csv** を再度実行して出力を確認します。

```
# oc get csv
```

出力例

```
NAME                                DISPLAY                    VERSION   REPLACES
PHASE
performance-addon-operator.v4.7.0   Performance Addon Operator   4.7.0     performance-
addon-operator.v4.6.0   Succeeded
```

16.4. リアルタイムおよび低レイテンシーワークロードのプロビジョニング

多くの企業や組織は、非常に高性能なコンピューティングを必要としており、とくに金融業界や通信業界では、低い、予測可能なレイテンシーが必要になる場合があります。このような固有の要件を持つ業界では、OpenShift Container Platform は Performance Addon Operator を提供して、OpenShift Container Platform アプリケーションの低レイテンシーのパフォーマンスと一貫性のある応答時間を実現するための自動チューニングを実装します。

クラスター管理者は、このパフォーマンスプロファイル設定を使用することにより、より信頼性の高い方法でこれらの変更を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 (リアルタイム)、ハウスキーピング用に予約される CPU、ワークロードの実行に使用される CPU に更新するかどうかを指定できます。

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お勧めします。

16.4.1. リアルタイムの既知の制限

注記

RT カーネルはワーカーノードでのみサポートされます。

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。

16.4.2. リアルタイム機能のあるワーカーのプロビジョニング

1. Performance Addon Operator をクラスターにインストールします。
2. オプション: ノードを OpenShift Container Platform クラスターに追加します。BIOS パラメーターの設定 について参照してください。
3. **oc**コマンドを使用して、リアルタイム機能を必要とするワーカーノードにラベル**worker-rt**を追加します。
4. リアルタイムノード用の新しいマシン設定プールを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-rt
  labels:
```

```
    machineconfiguration.openshift.io/role: worker-rt
spec:
  machineConfigSelector:
    matchExpressions:
      - {
          key: machineconfiguration.openshift.io/role,
          operator: In,
          values: [worker, worker-rt],
        }
  paused: false
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-rt: ""
```

マシン設定プール worker-rt は、**worker-rt** というラベルを持つノードのグループに対して作成されることに注意してください。

5. ノードロールラベルを使用して、ノードを適切なマシン設定プールに追加します。

注記

リアルタイムワークロードで設定するノードを決定する必要があります。クラスター内のすべてのノード、またはノードのサブセットを設定できます。Performance Addon Operator は、すべてのノードが専用のマシン設定プールの一部であることを想定します。すべてのノードを使用する場合は、Performance Addon Operator がワーカーノードのロールラベルを指すようにする必要があります。サブセットを使用する場合、ノードを新規のマシン設定プールにグループ化する必要があります。

6. ハウスキーピングコアの適切なセットと **realTimeKernel: enabled: true** を設定して **PerformanceProfile** を作成します。

7. **PerformanceProfile** で **machineConfigPoolSelector** を設定する必要があります:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: example-performanceprofile
spec:
...
  realTimeKernel:
    enabled: true
  nodeSelector:
     node-role.kubernetes.io/worker-rt: ""
  machineConfigPoolSelector:
     machineconfiguration.openshift.io/role: worker-rt
```

8. 一致するマシン設定プールがラベルを持つことを確認します。

```
$ oc describe mcp/worker-rt
```

出力例

```
Name:         worker-rt
Namespace:
Labels:       machineconfiguration.openshift.io/role=worker-rt
```

9. OpenShift Container Platform はノードの設定を開始しますが、これにより複数の再起動が伴う可能性があります。ノードが起動し、安定するのを待機します。特定のハードウェアの場合に、これには長い時間がかかる可能性がありますが、ノードごとに 20 分の時間がかかることが予想されます。

10. すべてが予想通りに機能していることを確認します。

16.4.3. リアルタイムカーネルのインストールの確認

以下のコマンドを使用して、リアルタイムカーネルがインストールされていることを確認します。

```
$ oc get node -o wide
```

文字列 **4.18.0-211.rt5.23.el8.x86_64** が含まれる、ロール **worker-rt** を持つワーカーに留意してください。

```
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   v1.22.1
128.66.135.107   <none>           Red Hat Enterprise Linux CoreOS 46.82.202008252340-0 (Ootpa)
4.18.0-211.rt5.23.el8.x86_64   cri-o://1.20.0-90.rhaos4.7.git4a0ac05.el8-rc.1
[...]
```

16.4.4. リアルタイムで機能するワークロードの作成

リアルタイム機能を使用するワークロードを準備するには、以下の手順を使用します。

手順

1. QoS クラスの **Guaranteed** を指定して Pod を作成します。
2. オプション: DPDK の CPU 負荷分散を無効にします。
3. 適切なノードセレクターを割り当てます。

アプリケーションを作成する場合には、アプリケーションのチューニングとデプロイメント に記載されている一般的な推奨事項に従ってください。

16.4.5. QoS クラスの Guaranteed を指定した Pod の作成

QoS クラスの **Guaranteed** が指定されている Pod を作成する際には、以下を考慮してください。

- Pod のすべてのコンテナーにはメモリー制限およびメモリー要求があり、それらは同じである必要があります。
- Pod のすべてのコンテナーには CPU の制限と CPU 要求が必要であり、それらは同じである必要があります。

以下の例は、1 つのコンテナーを持つ Pod の設定ファイルを示しています。コンテナーにはメモリー制限とメモリー要求があり、どちらも 200 MiB に相当します。コンテナーには CPU 制限と CPU 要求があり、どちらも 1 CPU に相当します。

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: <image-pull-spec>
    resources:
      limits:
        memory: "200Mi"
        cpu: "1"
      requests:
        memory: "200Mi"
        cpu: "1"
```

1. Pod を作成します。

```
$ oc  apply -f qos-pod.yaml --namespace=qos-example
```

2. Pod についての詳細情報を表示します。

```
$ oc get pod qos-demo --namespace=qos-example --output=yaml
```

出力例

```
spec:
  containers:
    ...
status:
  qosClass: Guaranteed
```

注記

コンテナーが独自のメモリー制限を指定するものの、メモリー要求を指定しない場合、OpenShift Container Platform は制限に一致するメモリー要求を自動的に割り当てます。同様に、コンテナーが独自の CPU 制限を指定するものの、CPU 要求を指定しない場合、OpenShift Container Platform は制限に一致する CPU 要求を自動的に割り当てます。

16.4.6. オプション: DPDK 用の CPU 負荷分散の無効化

CPU 負荷分散を無効または有効にする機能は CRI-O レベルで実装されます。CRI-O のコードは、以下の要件を満たす場合にのみ CPU の負荷分散を無効または有効にします。

- Pod は **performance-<profile-name>** ランタイムクラスを使用する必要があります。以下に示すように、パフォーマンスプロファイルのステータスを確認して、適切な名前を取得できます。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
...
status:
  ...
  runtimeClass: performance-manual
```

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

Performance Addon Operator は、該当するノードで高パフォーマンスのランタイムハンドラー設定スニペットの作成や、クラスターで高パフォーマンスのランタイムクラスの作成を行います。これには、CPU 負荷分散の設定機能を有効にすることを除くと、デフォルトのランタイムハンドラーと同じ内容が含まれます。

Pod の CPU 負荷分散を無効にするには、 **Pod** 仕様に以下のフィールドが含まれる必要があります。

```
apiVersion: v1
kind: Pod
metadata:
  ...
  annotations:
    ...
    cpu-load-balancing.crio.io: "disable"
    ...
  ...
spec:
  ...
  runtimeClassName: performance-<profile_name>
  ...
```

注記

CPU マネージャーの静的ポリシーが有効にされている場合に、CPU 全体を使用する Guaranteed QoS を持つ Pod について CPU 負荷分散を無効にします。これ以外の場合に CPU 負荷分散を無効にすると、クラスター内の他のコンテナーのパフォーマンスに影響する可能性があります。

16.4.7. 適切なノードセレクターの割り当て

Pod をノードに割り当てる方法として、以下に示すようにパフォーマンスプロファイルが使用するものと同じノードセレクターを使用することが推奨されます。

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  # ...
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
```

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

16.4.8. リアルタイム機能を備えたワーカーへのワークロードのスケジューリング

Performance Addon Operator によって低レイテンシーを確保するために設定されたマシン設定プールに割り当てられるノードに一致するラベルセレクターを使用します。詳細は、Assigning pods to nodes を参照してください。

16.4.9. Huge Page の設定

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

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

たとえば、パフォーマンスプロファイルの **hugepages pages** セクションで、**size**、**count**、およびオプションで **node** の複数のブロックを指定できます。

```
hugepages:
  defaultHugepagesSize: "1G"
  pages:
  - size:  "1G"
    count:  4
    node:  0 1
```

1 **node** は、Huge Page が割り当てられる NUMA ノードです。 **node** を省略すると、ページはすべての NUMA ノード間で均等に分散されます。

注記

更新が完了したことを示す関連するマシン設定プールのステータスを待機します。

これらは、Huge Page を割り当てるのに必要な唯一の設定手順です。

検証

- 設定を確認するには、ノード上の **/proc/meminfo** ファイルを参照します。

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

```
# grep -i huge /proc/meminfo
```

出力例

```
AnonHugePages:    ###### ##
ShmemHugePages:        0 kB
HugePages_Total:       2
HugePages_Free:        2
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       #### ##
Hugetlb:            #### ##
```

- 新規サイズを報告するには、**oc describe** を使用します。

```
$ oc describe node worker-0.ocp4poc.example.com | grep -i huge
```

出力例

```
                                   hugepages-1g=true
 hugepages-###:  ###
 hugepages-###:  ###
```

16.4.10. 複数の Huge Page サイズの割り当て

同じコンテナーで異なるサイズの Huge Page を要求できます。これにより、Huge Page サイズのニーズの異なる複数のコンテナーで設定されるより複雑な Pod を定義できます。

たとえば、サイズ **1G** と **2M** を定義でき、Performance Addon Operator は以下に示すようにノード上に両方のサイズを設定できます。

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

16.5. INFRA およびアプリケーションコンテナーの CPU の制限

一般的なハウスキーピングおよびワークロードタスクは、レイテンシーの影響を受けやすいプロセスに影響を与える可能性のある方法で CPU を使用します。デフォルトでは、コンテナーランタイムはすべてのオンライン CPU を使用して、すべてのコンテナーを一緒に実行します。これが原因で、コンテキストスイッチおよびレイテンシーが急増する可能性があります。CPU をパーティション化することで、ノイズの多いプロセスとレイテンシーの影響を受けやすいプロセスを分離し、干渉を防ぐことができます。以下の表は、Performance Add-On Operator を使用してノードを調整した後、CPU でプロセスがどのように実行されるかを示しています。

表16.1 プロセスの CPU 割り当て

プロセスタイプ	Details
Burstable および **BestEffort** Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
インフラストラクチャー Pod	低レイテンシーのワークロードが実行されている場合を除き、任意の CPU で実行されます。
割り込み	予約済み CPU にリダイレクトします (OpenShift Container Platform 4.7 以降ではオプション)

プロセスタイプ	Details
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。4.7 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。

16.6. 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。

手順

1. クラスターを準備します。

2. マシン設定プールを作成します。

3. Performance Addon Operator をインストールします。

4. 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。パフォーマンスプロファイルでは、カーネルを kernel-rt、hugepages の割り当て、オペレーティングシステムのハウスキーピング用に予約される CPU、およびワークロードの実行に使用される CPU に更新するかどうかを指定できます。
これは、典型的なパフォーマンスプロファイルです。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:
  isolated: "5-15"
  reserved: "0-4"
 hugepages:
  defaultHugepagesSize: "1G"
  pages:
  -size: "1G"
   count: 16
   node: 0
 realTimeKernel:
  enabled: true 1
 numa: 2
  topologyPolicy: "best-effort"
 nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
```

1 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。

2 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。

16.6.1. CPU のパーティション設定

単一の NUMA ノードからオペレーティングシステムのハウスキーピングタスク用にコアまたはスレッドを予約し、ワークロードを別の NUMA ノードに配置することができます。これを実行する理由として、ハウスキーピングプロセスで同じ CPU 上で実行されるレイテンシーの影響を受けるプロセスに影響を与える可能性がある方法で CPU を使用する可能性があるためです。ワークロードを別の NUMA ノードで維持することにより、プロセスが相互に干渉しないようにすることができます。また、各 NUMA ノードには共有されない独自のメモリーバスがあります。

spec セクションに、CPU の 2 つのグループを指定します。

- **isolated**: 最も低いレイテンシーが設定されます。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**: ハウスキーピング CPU予約されたグループのスレッドはビジーになる可能性があるため、レイテンシーの影響を受けるアプリケーションは分離されたグループで実行される必要があります。Create a pod that gets assigned a QoS class of **Guaranteed** を参照してください。

16.7. プラットフォーム検証のエンドツーエンドテストの実行

Cloud-native Network Functions (CNF) テストイメージは、CNF ペイロードの実行に必要な機能を検証するコンテナー化されたテストスイートです。この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 クラスターを検証できます。

イメージで実行されるテストは、3 つの異なるフェーズに分かれます。

- 単純なクラスター検証
- セットアップ
- エンドツーエンドテスト

検証フェーズでは、テストに必要なすべての機能がクラスターに正しくデプロイされていることを確認します。

検証には以下が含まれます。

- テストするマシンに属するマシン設定プールのターゲット設定
- ノードでの SCTP の有効化
- マシン設定を使用した xt_u32 カーネルモジュールの有効化
- Performance Addon Operator がインストールされていること
- SR-IOV Operator がインストールされていること
- PTP Operator がインストールされていること
- OVN kubernetes の SDN としての使用

CNF-test コンテナーの一部であるレイテンシーテストでも、同じ検証が必要になります。レイテンシーテストの実行に関する詳細は、レイテンシーテストの実行セクションを参照してください。

テストは、実行されるたびに環境設定を実行する必要があります。これには、SR-IOV ノードポリシー、パフォーマンスプロファイル、または PTP プロファイルの作成などの項目が関係します。テストですでに設定されているクラスターを設定できるようにすると、クラスターの機能が影響を受ける可能性があります。また、SR-IOV ノードポリシーなどの設定項目への変更により、設定の変更が処理されるまで環境が一時的に利用できなくなる可能性があります。

16.7.1. 前提条件

- テストエントリーポイントは **/usr/bin/test-run.sh** です。設定されたテストセットと実際の適合テストスイートの両方を実行します。最小要件として、これをボリューム経由でマウントされる kubeconfig ファイルおよび関連する **$KUBECONFIG** 環境変数と共に指定します。
- このテストでは、特定の機能が Operator、クラスターで有効にされるフラグ、またはマシン設定の形式でクラスターで利用可能であることを前提としています。
- テストによっては、変更を追加するための既存のマシン設定プールが必要です。これは、テストを実行する前にクラスター上に作成する必要があります。
  デフォルトのワーカープールは **worker-cnf** であり、以下のマニフェストを使用して作成できます。

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

ROLE_WORKER_CNF 変数を使用して、ワーカープール名を上書きできます。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
ROLE_WORKER_CNF=custom-worker-pool registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7
/usr/bin/test-run.sh
```

注記

現時点で、プールに属するノードですべてのテストが選択的に実行される訳ではありません。

16.7.2. テストの実行

kubeconfig ファイルが現在のフォルダーにある場合、テストスイートを実行するコマンドは以下のようになります。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
```

これにより、**kubeconfig** ファイルを実行中のコンテナー内から使用できます。

16.7.2.1. レイテンシーテストの実行

OpenShift Container Platform 4.7 では、CNF-test コンテナーからレイテンシーテストを実行することもできます。レイテンシーテストでは、パフォーマンス、スループットおよびレイテンシーを判別できるようにレイテンシーの制限を設定することができます。

レイテンシーテストでは **oslat** ツールを実行します。これは、OS レベルのレイテンシーを検知するオープンソースプログラムです。詳細は、Red Hat ナレッジベースソリューションの How to measure OS and hardware latency on isolated CPU? を参照してください。

デフォルトで、レイテンシーテストは無効になります。レイテンシーテストを有効にするには、**LATENCY_TEST_RUN** 変数を追加し、その値を **true** に設定する必要があります。例: **LATENCY_TEST_RUN=true**

さらに、レイテンシーテストに以下の環境変数を設定することもできます。

- **LATENCY_TEST_RUNTIME**: レイテンシーテストの実行に必要な時間 (秒単位) を指定します。
- **OSLAT_MAXIMUM_LATENCY**: **oslat** テストの実行時にすべてのバケットで予想される最大レイテンシー (マイクロ秒単位) を指定します。

レイテンシーテストを実行するには、以下のコマンドを実行します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e
OSLAT_MAXIMUM_LATENCY=20 registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-
run.sh
```

注記

レイテンシーテストは検出モードで実行する必要があります。詳細は、検出モードのセクションを参照してください。

以下のコマンドの使用による 10 秒のレイテンシーテストの結果サンプルの抜粋:

```
[root@cnf12-installer ~]# podman run --rm -v $KUBECONFIG:/kubeconfig:Z -e
PERF_TEST_PROFILE=worker-cnf-2 -e KUBECONFIG=/kubeconfig -e LATENCY_TEST_RUN=true
-e LATENCY_TEST_RUNTIME=10 -e OSLAT_MAXIMUM_LATENCY=20 -e
DISCOVERY_MODE=true registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
-ginkgo.focus="Latency"
running /0_config.test -ginkgo.focus=Latency
```

出力例

```
I1106 15:09:08.087085       7 request.go:621] Throttling request took 1.037172581s, request:
GET:https://api.cnf12.kni.lab.eng.bos.redhat.com:6443/apis/autoscaling.openshift.io/v1?timeout=32s
Running Suite: Performance Addon Operator configuration

Random Seed: 1604675347
Will run 0 of 1 specs

JUnit report was created: /unit_report_performance_config.xml

Ran 0 of 1 Specs in 0.000 seconds
SUCCESS! -- 0 Passed | 0 Failed | 0 Pending | 1 Skipped
PASS
running /4_latency.test -ginkgo.focus=Latency
I1106 15:09:10.735795      23 request.go:621] Throttling request took 1.037276624s, request:
GET:https://api.cnf12.kni.lab.eng.bos.redhat.com:6443/apis/certificates.k8s.io/v1?timeout=32s
Running Suite: Performance Addon Operator latency e2e tests

Random Seed: 1604675349
Will run 1 of 1 specs

I1106 15:10:06.401180      23 nodes.go:86] found mcd machine-config-daemon-r78qc for node
cnfdd8.clus2.t5g.lab.eng.bos.redhat.com
I1106 15:10:06.738120      23 utils.go:23] run command 'oc [exec -i -n openshift-machine-config-
operator -c machine-config-daemon --request-timeout 30 machine-config-daemon-r78qc -- cat
```

```
/rootfs/var/log/oslat.log]' (err=<nil>):
  stdout=
Version: v0.1.7

Total runtime:   10 seconds
Thread priority:  SCHED_FIFO:1
CPU list:
3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,3
8,39,40,41,42,43,44,45,46,47,48,49,50
CPU for main thread:  2
Workload:   no
Workload mem:   0 (KiB)
Preheat cores:   48

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

Core: 3 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
CPU Freq: 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2096 2092 2096 2096
2096 2092 2092 2096 2096 2096 2096 2096 2096 2096 2096 2096 2092 2096 2096 2092 2096
2096 2096 2096 2092 2096 2096 2096 2092 2096 2096 2096 2096 2096 2096 (Mhz)
...
Maximum: 3 4 3 3 3 3 3 3 4 3 3 3 3 4 3 3 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 4 3 3 3 3 3 4 3 3 3 4
(us)
```

16.7.3. イメージパラメーター

要件に応じて、テストは異なるイメージを使用できます。以下の環境変数を使用して変更できるテストで、以下の 2 つのイメージが使用されます。

- **CNF_TESTS_IMAGE**
- **DPDK_TESTS_IMAGE**

たとえば、カスタムレジストリーを使用して **CNF_TESTS_IMAGE** を変更するには、以下のコマンドを実行します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
CNF_TESTS_IMAGE="custom-cnf-tests-image:latests" registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.7 /usr/bin/test-run.sh
```

16.7.3.1. ginkgo パラメーター

テストスイートは、ginkgo BDD フレームワーク上に構築されます。これは、テストをフィルターするか、または省略するためのパラメーターを受け入れることを意味します。

-ginkgo.focus パラメーターを使用してテストセットをフィルターできます。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.focus="performance|sctp"
```

-ginkgo.focus パラメーターを使用してのみ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のみを実行するには、テストする必要があるパフォーマンスプロファイルの名前が含まれる **-ginkgo.focus** パラメーターと **PERF_TEST_PROFILE** 環境変数を指定する必要があります。以下に例を示します。

```
$ docker run --rm -v $KUBECONFIG:/kubeconfig -e KUBECONFIG=/kubeconfig -e
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e
OSLAT_MAXIMUM_LATENCY=20 -e PERF_TEST_PROFILE=<performance_profile_name>
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.focus="\[performance\]\
[config\]|\[performance\]\ Latency\ Test"
```

注記

特定のテストでは、SR-IOV と SCTP の両方が必要になります。**focus** パラメーターの選択的な性質を考慮すると、このテストは **sriov** Matcher のみを配置してトリガーできます。テストが SR-IOV がインストールされているクラスターに対して実行されるものの、SCTP がはない場合、**-ginkgo.skip=SCTP** パラメーターを追加すると、テストは SCTP テストを省略します。

16.7.3.2. 利用可能な機能

フィルターに使用できる機能のセットは以下になります。

- **performance**
- **sriov**
- **ptp**
- **sctp**
- **xt_u32**
- **dpdk**

16.7.4. ドライラン

このコマンドを使用してドライランモードで実行します。これは、テストスイートの内容を確認し、イメージが実行するすべてのテストの出力を提供します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh -ginkgo.dryRun -ginkgo.v
```

16.7.5. 非接続モード

CNF は、非接続クラスター、つまり外部レジストリーに到達できないクラスターでテストを実行してイメージのサポートをテストします。これは 2 つの手順で行います。

1. ミラーリングの実行。
2. テストに対してカスタムレジストリーからのイメージを使用するように指示します。

16.7.5.1. 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 実行可能ファイルはイメージに含まれ、テストをローカルレジストリーに対して実行するために必要なテストをミラーリングするために **oc** で必要な入力を提供します。

クラスターおよびインターネット経由で registry.redhat.io にアクセスできる中間マシンから、以下のコマンドを実行します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/mirror -registry my.local.registry:5000/ |  oc
image mirror -f -
```

次に、イメージの取得に使用されるレジストリーを上書きする方法について、以下のセクションの手順に従います。

16.7.5.2. テストに対してカスタムレジストリーからのそれらのイメージを使用するように指示します。

これは、**IMAGE_REGISTRY** 環境変数を設定して実行されます。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
IMAGE_REGISTRY="my.local.registry:5000/" -e CNF_TESTS_IMAGE="custom-cnf-tests-
image:latests" registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
```

16.7.5.3. クラスター内部レジストリーへのミラーリング

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. 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 **cnftests** 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。

   ```
   $ oc policy add-role-to-user system:image-puller system:serviceaccount:sctptest:default --
   namespace=cnftests
   ```

   ```
   $ 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
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:dpdk-testing:default --namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:sriov-conformance-testing:default --namespace=cnftests
```

5. 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. 以下のような **dockerauth.json** を作成します。

```
echo "{\"auths\": { \"$REGISTRY\": { \"auth\": $TOKEN } }}" > dockerauth.json
```

7. ミラーリングを実行します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /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。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests cnf-tests-local:latest /usr/bin/test-run.sh
```

16.7.5.4. イメージの異なるセットのミラーリング

手順

1. **mirror** コマンドは、デフォルトで u/s イメージのミラーリングを試行します。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:4.7"
    },
    {
        "registry": "public.registry.io:5000",
        "image": "imagefordpdk:4.7"
    }
]
```

2. これを **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できます。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。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/mirror --registry
"my.local.registry:5000/" --images "/kubeconfig/images.json" |  oc image mirror -f -
```

16.7.6. 検出モード

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

一部の設定項目は、テストで引き続き作成されます。これらの項目は、テストを実行するために必要な特定の項目です (例: SR-IOV ネットワーク)。これらの設定項目はカスタム namespace に作成され、テストの実行後にクリーンアップされます。

さらに、これによりテストの実行時間が削減されます。設定項目はすでに存在しているので、環境設定および安定化に追加の時間は取る必要はありません。

検出モードを有効にするには、以下のように **DISCOVERY_MODE** 環境変数を設定してテストに対して指示する必要があります。

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
DISCOVERY_MODE=true registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-run.sh
```

16.7.6.1. 必要な環境設定の前提条件

SR-IOV テスト

ほとんどの SR-IOV テストには以下のリソースが必要です。

- **SriovNetworkNodePolicy**
- **SriovNetworkNodePolicy** で指定されるリソースの 1 つ以上が割り当て可能な状態であること。リソース数が 5 以上の場合に十分であると見なされます。

テストによっては、追加の要件があります。

- 利用可能なポリシーリソースがあるノード上の未使用のデバイス。リンク状態が **DOWN** であり、ブリッジスレーブではない。
- MTU 値が **9000** の **SriovNetworkNodePolicy**。

DPDK テスト

DPDK 関連のテストには、以下が必要です。

- パフォーマンスプロファイル
- SR-IOV ポリシー。
- SRIOV ポリシーで利用可能なリソースが含まれ、**PerformanceProfile** ノードセレクターで利用可能なノード。

PTP テスト

- スレーブ **PtpConfig** (**ptp4lOpts="-s" ,phc2sysOpts="-a -r"**)。
- スレーブ **PtpConfig** に一致するラベルの付いたノード。

SCTP テスト

- **SriovNetworkNodePolicy**
- **SriovNetworkNodePolicy** および SCTP を有効にする **MachineConfig** の両方に一致するノード。

XT_U32 テスト

- XT_U32 を有効にするマシン設定を持つノード。

Performance Operator のテスト

各種のテストにはそれぞれ異なる要件があります。以下はその一部になります。

- パフォーマンスプロファイル
- **profile.Spec.CPU.Isolated = 1** を持つパフォーマンスプロファイル。
- **profile.Spec.RealTimeKernel.Enabled == true** を持つパーマンスプロファイル。
- Huge Page が使用されていないノード。

16.7.6.2. テスト中に使用されるノードの制限

テストが実行されるノードは、**NODES_SELECTOR** 環境変数を指定して制限できます。テストによって作成されるリソースは、指定されるノードに制限されます。

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
NODES_SELECTOR=node-role.kubernetes.io/worker-cnf registry.redhat.io/openshift-kni/cnf-tests
/usr/bin/test-run.sh
```

16.7.6.3. 単一のパフォーマンスプロファイルの使用

DPDK テストで必要なリソースは、パフォーマンスのテストスイートで必要なリソースのレベルよりも高くなります。実行をより高速にするために、テストで使用されるパフォーマンスプロファイルは、DPDK テストスイートも提供するプロファイルを使用して上書きできます。

これを実行するには、コンテナー内にマウントできる以下のようなプロファイルを使用し、パフォーマンステストに対してこれをデプロイするように指示できます。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "4-15"
    reserved: "0-3"
  hugepages:
    defaultHugepagesSize: "1G"
```

```
    pages:
    - size: "1G"
      count: 16
      node: 0
  realTimeKernel:
    enabled: true
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

使用されるパフォーマンスプロファイルを上書きするには、マニフェストをコンテナー内にマウントし、**PERFORMANCE_PROFILE_MANIFEST_OVERRIDE** パラメーターを設定してテストに対して指示する必要があります。

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubeconfig/manifest.yaml
registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-run.sh
```

16.7.6.4. パフォーマンスプロファイルのクリーンアップの無効化

検出モードで実行されない場合、スイートは作成されるすべてのアーティファクトおよび設定をクリーンアップします。これには、パフォーマンスプロファイルが含まれます。

パフォーマンスプロファイルを削除する際に、マシン設定プールが変更され、ノードが再起動されます。新規の繰り返しの後に、新規プロファイルが作成されます。これにより、長いテストサイクルが実行間に生じます。

このプロセスを迅速化するには、**CLEAN_PERFORMANCE_PROFILE="false"** を設定し、パフォーマンスプロファイルをクリーンアップしないようにテストに指示します。これにより、次の反復でこれを作成し、これが適用されるのを待機する必要がなくなります。

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
CLEAN_PERFORMANCE_PROFILE="false" registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-
run.sh
```

16.7.7. トラブルシューティング

クラスターはコンテナー内からアクセスできる必要があります。これを確認するには、以下を実行します。

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift-kni/cnf-tests oc get nodes
```

これが機能しない場合は、原因が DNS、MTU サイズ、またはファイアウォールの問題に関連している可能性があります。

16.7.8. テストレポート

CNF エンドツーエンドテストにより、JUnit テスト出力とテスト失敗レポートという 2 つの出力が生成されます。

16.7.8.1. JUnit テスト出力

レポートがダンプされるパスと共に **--junit** パラメーターを渡すと、JUnit 準拠の XML が作成されます。

```
$ docker run -v $(pwd)/:/kubeconfig -v $(pwd)/junitdest:/path/to/junit -e
KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-
run.sh --junit /path/to/junit
```

16.7.8.2. テスト失敗レポート

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が含まれるレポートは、レポートがダンプされるパスと共に **--report** パラメーターを渡すことで生成できます。

```
$ docker run -v $(pwd)/:/kubeconfig -v $(pwd)/reportdest:/path/to/report -e
KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-
run.sh --report /path/to/report
```

16.7.8.3. podman に関する注記

非 root として、また権限なしで podman を実行する場合、permission denied というエラーを出してパスのマウントに失敗する可能性があります。これを機能させるには、**:Z** をボリューム作成に追加します。たとえば、**-v $(pwd)/:/kubeconfig:Z** を使用して podman が SELinux のラベルを適切に書き換えることができます。

16.7.8.4. OpenShift Container Platform 4.4 での実行

以下を除き、CNF のエンドツーエンドのテストは OpenShift Container Platform 4.4 と互換性があります。

```
[test_id:28466][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration
injected through openshift-node-performance profile
[test_id:28467][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration
injected through the openshift-node-performance profile
```

これらのテストを省略するには、**-ginkgo.skip "28466|28467"** パラメーターを追加します。

16.7.8.5. 単一のパフォーマンスプロファイルの使用

DPDK テストには、パフォーマンステストスイートに必要なリソースよりも多くのリソースが必要です。実行をより迅速にするには、DPDK テストスイートを提供するプロファイルを使用して、テストが使用するパフォーマンスプロファイルを上書きすることができます。

これを実行するには、コンテナー内にマウントできる以下のようなプロファイルを使用し、パフォーマンステストに対してこれをデプロイするように指示できます。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:
  isolated: "5-15"
  reserved: "0-4"
 hugepages:
```

```
   defaultHugepagesSize: "1G"
   pages:
   -size: "1G"
    count: 16
    node: 0
  realTimeKernel:
   enabled: true
  numa:
   topologyPolicy: "best-effort"
  nodeSelector:
   node-role.kubernetes.io/worker-cnf: ""
```

パフォーマンスプロファイルを上書きするには、マニフェストをコンテナー内にマウントし、**PERFORMANCE_PROFILE_MANIFEST_OVERRIDE** を設定してテストに対して指示する必要があります。

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubeconfig/manifest.yaml
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.7 /usr/bin/test-run.sh
```

16.7.9. クラスターへの影響

機能によっては、テストスイートを実行すると、クラスターに異なる影響が及ぶ可能性があります。通常、SCTP テストのみではクラスター設定を変更しません。他のすべての機能は設定にさまざまな影響を与えます。

16.7.9.1. SCTP

SCTP テストは、接続性をチェックするために異なるノードで異なる Pod を実行するのみです。クラスターへの影響は、2 つのノードでの単純な Pod の実行に関連します。

16.7.9.2. XT_U32

XT_U32 テストは異なるノードで Pod を実行し、xt_u32 を使用する iptables ルールをチェックします。クラスターへの影響は、2 つのノードでの単純な Pod の実行に関連します。

16.7.9.3. SR-IOV

SR-IOV テストには、SR-IOV ネットワーク設定での変更が必要になります。この場合、テストは異なるタイプの設定を作成し、破棄します。

これは、既存の SR-IOV ネットワーク設定がクラスターにインストールされている場合に影響を与える可能性があります。これらの設定の優先順位によっては競合が生じる可能性があるためです。

同時に、テストの結果は既存の設定による影響を受ける可能性があります。

16.7.9.4. PTP

PTP テストは、PTP 設定をクラスターのノードセットに適用します。SR-IOV と同様に、これはすでに有効な既存の PTP 設定と競合する可能性があります。これにより予測不可能な結果が出される可能性があります。

16.7.9.5. パフォーマンス

パフォーマンステストにより、パフォーマンスプロファイルがクラスターに適用されます。これにより、ノード設定の変更、CPU の予約、メモリー Huge Page の割り当て、およびカーネルパッケージのリアルタイムの設定が行われます。**performance** という名前の既存のプロファイルがクラスターですでに利用可能な場合、テストはこれをデプロイしません。

16.7.9.6. DPDK

DPDK はパフォーマンスおよび SR-IOV 機能の両方に依存するため、テストスイートはパフォーマンスプロファイルと SR-IOV ネットワークの両方を設定します。そのため、これによる影響は SR-IOV テストおよびパフォーマンステストで説明されているものと同じになります。

16.7.9.7. クリーンアップ

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。

16.8. 低レイテンシー CNF チューニングステータスのデバッグ

PerformanceProfile カスタムリソース (CR) には、チューニングのステータスを報告し、レイテンシーのパフォーマンスの低下の問題をデバッグするためのステータスフィールドが含まれます。これらのフィールドは、Operator の調整機能の状態を記述する状態について報告します。

パフォーマンスプロファイルに割り当てられるマシン設定プールのステータスが degraded 状態になると典型的な問題が発生する可能性があり、これにより **PerformanceProfile** のステータスが低下します。この場合、マシン設定プールは失敗メッセージを発行します。

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

```
Status:
  Conditions:
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                True
    Type:                  Available
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                True
    Type:                  Upgradeable
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                False
    Type:                  Progressing
    Last Heartbeat Time:   2020-06-02T10:01:24Z
    Last Transition Time:  2020-06-02T10:01:24Z
    Status:                False
    Type:                  Degraded
```

Status フィールドには、 パフォーマンスプロファイルのステータスを示す **Type** 値を指定する **Conditions** が含まれます。

Available

すべてのマシン設定および Tuned プロファイルが正常に作成され、クラスターコンポーネントで利用可能になり、それら (NTO、MCO、Kubelet) を処理します。

Upgradeable

Operator によって維持されるリソースは、アップグレードを実行する際に安全な状態にあるかどうかを示します。

Progressing

パフォーマンスプロファイルからのデプロイメントプロセスが開始されたことを示します。

Degraded

以下の場合にエラーを示します。

- パーマンスプロファイルの検証に失敗しました。
- すべての関連するコンポーネントの作成が完了しませんでした。

これらのタイプには、それぞれ以下のフィールドが含まれます。

Status

特定のタイプの状態 (**true** または **false**)。

Timestamp

トランザクションのタイムスタンプ。

Reason string

マシンの読み取り可能な理由。

Message string

状態とエラーの詳細を説明する人が判読できる理由 (ある場合)。

16.8.1. マシン設定プール

パフォーマンスプロファイルとその作成される製品は、関連付けられたマシン設定プール (MCP) に従ってノードに適用されます。MCP は、カーネル引数、kube 設定、Huge Page の割り当て、および rt-kernel のデプロイメントを含むパフォーマンスアドオンが作成するマシン設定の適用についての進捗に関する貴重な情報を保持します。パフォーマンスアドオンコントローラーは MCP の変更を監視し、それに応じてパフォーマンスプロファイルのステータスを更新します。

MCP がパフォーマンスプロファイルのステータスに返す状態は、MCP が **Degraded** の場合のみとなり、この場合、**performaceProfile.status.condition.Degraded = true** になります。

例

以下の例は、これに作成された関連付けられたマシン設定プール (**worker-cnf**) を持つパフォーマンスプロファイルのサンプルです。

1. 関連付けられたマシン設定プールの状態は degraded (低下) になります。

```
# oc get mcp
```

出力例

```
NAME       CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master     rendered-master-2ee57a93fa6c9181b546ca46e1571d2d       True      False
False      3           3              3                0                    2d21h
worker     rendered-worker-d6b2bdc07d9f5a59a6b68950acf25e5f       True      False
```

```
False     2          2              2                0                  2d21h
worker-cnf   rendered-worker-cnf-6c838641b8a08fff08dbd8b02fb63f7c   False     True
True      2          1              1                1                  2d20h
```

2. MCP の **describe** セクションには理由が示されます。

```
# oc describe mcp worker-cnf
```

出力例

```
  Message:             Node node-worker-cnf is reporting: "prepping update:
  machineconfig.machineconfiguration.openshift.io \"rendered-worker-cnf-
40b9996919c08e335f3ff230ce1d170\" not
  found"
    Reason:              1 nodes are reporting degraded status on sync
```

3. degraded (低下) の状態は、**degraded = true** とマークされたパフォーマンスプロファイルの **status** フィールドにも表示されるはずです。

```
# oc describe performanceprofiles performance
```

出力例

```
Message: Machine config pool worker-cnf Degraded Reason: 1 nodes are reporting
degraded status on sync.
Machine config pool worker-cnf Degraded Message: Node yquinn-q8s5v-w-b-
z5lqn.c.openshift-gce-devel.internal is
reporting: "prepping update: machineconfig.machineconfiguration.openshift.io
\"rendered-worker-cnf-40b9996919c08e335f3ff230ce1d170\" not found".    Reason:
MCPDegraded
   Status:  True
   Type:    Degraded
```

16.9. 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。

16.9.1. 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。

16.9.2. 低レイテンシーチューニングデータの収集について

oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、以下を始めとする低レイテンシーチューニングに関連する機能およびオブジェクトが含まれます。

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

must-gatherを使用して Performance Addon Operator のデバッグ情報を収集するには、Performance Addon Operator の**must-gather**イメージを指定する必要があります。

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.7.
```

16.9.3. 特定の機能に関するデータ収集

oc adm must-gather CLI コマンドを **--image** または **--image-stream** 引数と共に使用して、特定に機能についてのデバッグ情報を収集できます。**must-gather** ツールは複数のイメージをサポートするため、単一のコマンドを実行して複数の機能についてのデータを収集できます。

> **注記**
>
> 特定の機能データに加えてデフォルトの **must-gather** データを収集するには、**--image-stream=openshift/must-gather** 引数を追加します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform CLI (oc) がインストールされている。

手順

1. **must-gather** データを保存するディレクトリーに移動します。
2. **oc adm must-gather** コマンドを 1 つまたは複数の **--image** または **--image-stream** 引数と共に実行します。たとえば、以下のコマンドは、デフォルトのクラスターデータと Performance Addon Operator に固有の情報の両方を収集します。

```
$ oc adm must-gather \
 --image-stream=openshift/must-gather \ 1

 --image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.7
2
```

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

2 低レイテンシーチューニングの診断用の **must-gather** イメージ。

3. 作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1 **must-gather-local.5421342344627712289/** を実際のディレクトリー名に置き換えます。

4. 圧縮ファイルを Red Hat カスタマーポータル で作成したサポートケースに添付します。

関連情報

- MachineConfig および KubeletConfig についての詳細は、ノードの管理 を参照してください。
- Node Tuning Operator についての詳細は、 Node Tuning Operator の使用 を参照してください。
- PerformanceProfile についての詳細は、Huge Page の設定 を参照してください。
- コンテナーからの Huge Page の消費に関する詳細は、Huge Page がアプリケーションによって消費される仕組み を参照してください。

第17章 INTEL VRAN DEDICATED ACCELERATOR ACC100 でのデータプレーンのパフォーマンスの最適化

17.1. VRAN DEDICATED ACCELERATOR ACC100 について

Intel 高速 4G/LTE および 5G Virtualized Radio Access Networks (vRAN) ワークロードからのハードウェアアクセラレーターカード。これにより、市販の既製のプラットフォームの全体的な計算能力が向上します。

Intel の eASIC テクノロジーに基づく vRAN Dedicated Accelerator ACC100 は、4G/LTE および 5G テクノロジーに対する前方誤り訂正 (FEC) のコンピューター集約型プロセスをオフロードし、処理能力を解放するように設計されています。Intel eASIC デバイスは設定された ASIC で、FPGA と標準のアプリケーション固有の統合サーキット (ASIC) 間の中間テクノロジーです。

OpenShift Container Platform での Intel vRAN Dedicated Accelerator ACC100 サポートは 1 つの Operator を使用します。

- OpenNESS Operator for Wireless FEC Accelerators

17.2. ワイヤレス FEC ACCELERATOR 向けの OPENNESS SR-IOV OPERATOR のインストール

Intel Wireless forward error correction (FEC) Accelerator 向けの OpenNESS Operator のロールは、OpenShift Container Platform クラスター内の Intel vRAN FEC Acceleration ハードウェアで公開されるデバイスをオーケストレーションして管理することです。

最もコンピューター集約型な 4G/LTE および 5G ワークロードの 1 つは、RAN レイヤー 1 (L1) FEC になります。FEC は、信頼性が低い通信チャネルまたはノイズ一通信チャネルでデータ転送エラーを解決します。FEC テクノロジーは、再送信を必要とせずに 4G/LTE または 5G データ内のエラーの一部を検出して修正します。

Intel vRAN Dedicated Accelerator ACC100 が提供する FEC デバイスは、vRAN のユースケースに対応しています。

ワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator は、FEC デバイスの仮想機能 (VF) を作成し、それらを適切なドライバーにバインドし、4G/LTE または 5G デプロイメントの機能について VF キューを設定します。

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して、ワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator をインストールできます。

17.2.1. CLI の使用によるワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator のインストール

クラスター管理者は、CLI を使用してワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator をインストールできます。

前提条件

- ベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) 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。

手順

1. 以下のアクションを実行して、ワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator の namespace を作成します。
   a. 以下の例のように **sriov-namespace.yaml** という名前のファイルを作成し、**vran-acceleration-operators** namespace を定義します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: vran-acceleration-operators
  labels:
    openshift.io/cluster-monitoring: "true"
```

   b. 以下のコマンドを実行して namespace を作成します。

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

2. 以下のオブジェクトを作成して、ワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator を直前の手順で作成した namespace にインストールします。
   a. 以下の **OperatorGroup** カスタムリソース (CR) を作成し、YAML を **sriov-operatorgroup.yaml** ファイルに保存します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: vran-operators
  namespace: vran-acceleration-operators
spec:
  targetNamespaces:
    - vran-acceleration-operators
```

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

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

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

```
$ oc get packagemanifest sriov-fec -n openshift-marketplace -o
jsonpath='{.status.defaultChannel}'
```

出力例

```
stable
```

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

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: sriov-fec-subscription
  namespace: vran-acceleration-operators
spec:
  channel: "<channel>" 1
  name: sriov-fec
  source: certified-operators 2
  sourceNamespace: openshift-marketplace
```

1. **.status.defaultChannel** パラメーターの直前の手順で取得した値からチャネルの値を指定します。

2. **certified-operators** 値を指定する必要があります。

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

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

検証

- Operator がインストールされていることを確認します。

```
$ oc get csv -n vran-acceleration-operators -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                  Phase
sriov-fec.v1.1.0        Succeeded
```

17.2.2. Web コンソールを使用したワイヤレス FEC Accelerator の OpenNESS SR-IOV 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ワイヤレス FEC Accelerator の OpenNESS SR-IOV Operator をインストールできます。

> **注記**
>
> 先のセクションで説明されているように **Namespace** および **OperatorGroup** カスタムリソース (CR) を作成する必要があります。

手順

1. OpenShift Container Platform Web コンソールを使用して、ワイヤレス FEC Accelerator の OpenNESS SR-IOV 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 の一覧から **ワイヤレス FEC Accelerator 向け OpenNESS SR-IOV Operator** を選択し、**Install** をクリックします。

c. **Install Operator** ページで、**All namespaces on the cluster**を選択します。次に、**Install** をクリックします。

2. オプション: SRIOV-FEC Operator が正常にインストールされていることを確認します。

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

b. **ワイヤレス FEC Accelerator 向け OpenNESS SR-IOV Operator**が **Status** が **InstallSucceeded** の **vran-acceleration-operators** プロジェクトに一覧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。

コンソールが Operator がインストールされていることを示していない場合は、以下のトラブルシューティング手順を実行します。

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

17.2.3. Intel® vRAN Dedicated Accelerator ACC100 向け SR-IOV FEC Operator の設定

Intel vRAN Dedicated Accelerator ACC100 のプログラミングは、vRAN ワークロードで前方誤り訂正 (FEC) を加速するのに使用される SR-IOV (Single Root I/O Virtualization) 仮想関数 (VF) デバイスを公開します。Intel vRAN Dedicated Accelerator ACC100 は 4G および 5G vRAN (Virtualized Radio Access Networks) ワークロードを加速します。これにより、市販の既製のプラットフォームの全体的な計算能力が向上します。このデバイスは Mount Bryce としても知られています。

SR-IOV-FEC Operator は、vRAN L1 アプリケーションの FEC プロセスを加速するために使用される FEC デバイスの管理を処理します。

SR-IOV-FEC Operator を設定するには、以下のことを行う必要があります。

- FEC デバイスの VF (Virtual Function) の作成
- VF を適切なドライバーにバインド
- 4G または 5G デプロイメントで希望する機能向け VF キューの設定

前方誤り訂正 (FEC) のロールは、メッセージ内の特定のビットが失われたり、文字化けしている可能性がある転送エラーの修正です。伝送メディアのノイズ、干渉、または信号強度の低下により、メッセージが失われたり文字化けしたりする可能性があります。FEC を使用しないと、文字化けしたメッセージは、ネットワーク負荷に加え、スループットとレイテンシーに影響を与える必要があります。

前提条件

- Intel FPGA ACC100 5G/4G カード。

- ワイヤレス FEC Accelerator 向け OpenNESS Operator でインストールされる 1 つまたは複数のノード。
- ノードの BIOS でグローバル SR-IOV および VT-d 設定を有効にします。
- Performance Addon Operator で設定された RT カーネル。
- **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. **vran-acceleration-operators** プロジェクトに切り替えます。

```
$ oc project vran-acceleration-operators
```

2. SR-IOV-FEC Operator がインストールされていることを確認します。

```
$ oc get csv -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                              Phase
sriov-fec.v1.1.0                    Succeeded
```

3. **sriov-fec** Pod が実行していることを確認します。

```
$  oc get pods
```

出力例

```
NAME                                    READY      STATUS      RESTARTS    AGE
sriov-device-plugin-j5jlv                    1/1        Running     1          15d
sriov-fec-controller-manager-85b6b8f4d4-gd2qg   1/1        Running     1          15d
sriov-fec-daemonset-kqqs6                      1/1        Running     1          15d
```

- **sriov-device-plugin** は、FEC 仮想機能をノードの下にあるリソースとして公開します。
- **sriov-fec-controller-manager** は CR をノードに適用し、オペランドコンテナーを維持します。
- **sriov-fec-daemonset** は、次のことを行います。
    - 各ノードで SRIOV NIC の検出
    - ステップ 6 で定義されたカスタムリソース (CR) のステータスの同期
    - CR の spec を入力として実行し、検出された NIC の設定

4. サポート対象の vRAN FEC アクセラレーターデバイスのいずれかを含むすべてのノードを取得します。

```
$ oc get sriovfecnodeconfig
```

出力例

```
NAME          CONFIGURED
node1        Succeeded
```

5. 設定する SR-IOV FEC アクセラレーターデバイスの Physical Function (PF) を検索します。

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

出力例

```
status:
    conditions:
    - lastTransitionTime: "2021-03-19T17:19:37Z"
      message: Configured successfully
      observedGeneration: 1
      reason: ConfigurationSucceeded
      status: "True"
      type: Configured
    inventory:
      sriovAccelerators:
      - deviceID: 0d5c
        driver: ""
        maxVirtualFunctions: 16
        pciAddress: 0000:af:00.0 (1)
        vendorID: "8086"
        virtualFunctions: [] (2)
```

1. このフィールドは、カードの PCI アドレスを表示します。

2. このフィールドは、仮想関数が空であることを示します。

6. FEC デバイスの仮想機能とキューグループの数を設定します。

    a. 以下のカスタムリソース (CR) を作成し、YAML を **sriovfec_acc100cr.yaml** ファイルに保存します。

> **注記**
>
> この例では、5G の ACC100 8/8 キューグループ、Uplink 向け 4 キューグループ、および Downlink の別の 4 つのキューグループを設定します。

```
apiVersion: sriovfec.intel.com/v1
kind: SriovFecClusterConfig
metadata:
  name: config (1)
spec:
  nodes:
   - nodeName: node1 (2)
     physicalFunctions:
       - pciAddress: 0000:af:00.0 (3)
         pfDriver: "pci-pf-stub"
         vfDriver: "vfio-pci"
         vfAmount: 16 (4)
```

```
bbDevConfig:
  acc100:
    # Programming mode: 0 = VF Programming, 1 = PF Programming
    pfMode: false
    numVfBundles: 16
    maxQueueSize: 1024
    uplink4G:
      numQueueGroups: 0
      numAqsPerGroups: 16
      aqDepthLog2: 4
    downlink4G:
     numQueueGroups: 0
     numAqsPerGroups: 16
     aqDepthLog2: 4
    uplink5G:
     numQueueGroups: 4
     numAqsPerGroups: 16
     aqDepthLog2: 4
    downlink5G:
     numQueueGroups: 4
     numAqsPerGroups: 16
     aqDepthLog2: 4
```

1. CR オブジェクトの名前を指定します。指定できる唯一の名前は **config** です。
2. ノード名を指定します。
3. SR-IOV-FEC Operator がインストールされているカードの PCI アドレスを指定します。
4. 作成する仮想関数の数を指定します。Intel vRAN Dedicated Accelerator ACC100 の場合は、16 個の VF をすべて作成します。

注記

カードは、グループごとに最大 16 個のキューを持つ最大 8 個のキューを提供するように設定されます。キューグループは、5G と 4G と Uplink と Downlink に割り当てられたグループ間で分割できます。Intel vRAN Dedicated Accelerator ACC100 を設定することができます。

- 4G または 5G のみ
- 4G と 5G を同時に

設定した各 VF は、すべてのキューにアクセスできます。各キューグループの優先度は、それぞれ個別の優先レベルを持ちます。特定のキューグループの要求はアプリケーションレベル (FEC デバイスを利用する vRAN アプリケーション) から行われます。

b. CR を適用します。

```
$ oc apply -f sriovfec_acc100cr.yaml
```

CR の適用後、SR-IOV FEC デーモンは FEC デバイスの設定を開始します。

検証

1. ステータスを確認します。

```
$ oc get sriovfecclusterconfig config -o yaml
```

出力例

```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T11:46:22Z"
    message: Configured successfully
    observedGeneration: 1
    reason: Succeeded
    status: "True"
    type: Configured
  inventory:
    sriovAccelerators:
    - deviceID: 0d5c
      driver: pci-pf-stub
      maxVirtualFunctions: 16
      pciAddress: 0000:af:00.0
      vendorID: "8086"
      virtualFunctions:
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.0
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.1
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.2
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.3
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.4
```

2. ログを確認します。

   a. SR-IOV デーモンの Pod 名を判別します。

   ```
   $ oc get po -o wide | grep sriov-fec-daemonset | grep node1
   ```

   出力例

   ```
   sriov-fec-daemonset-kqqs6                      1/1     Running   0          19h
   ```

   b. ログを表示します。

   ```
   $ oc logs sriov-fec-daemonset-kqqs6
   ```

出力例


```
{"level":"Level(-
2)","ts":1616794345.4786215,"logger":"daemon.drainhelper.cordonAndDrain()","msg":"no
de drained"}
{"level":"Level(-
4)","ts":1616794345.4786265,"logger":"daemon.drainhelper.Run()","msg":"worker
function - start"}
{"level":"Level(-
4)","ts":1616794345.5762916,"logger":"daemon.NodeConfigurator.applyConfig","msg":"cur
rent node status","inventory":{"sriovAccelerat
ors":
[{"vendorID":"8086","deviceID":"0b32","pciAddress":"0000:20:00.0","driver":"","maxVirtualF
unctions":1,"virtualFunctions":[]},{"vendorID":"8086"
,"deviceID":"0d5c","pciAddress":"0000:af:00.0","driver":"","maxVirtualFunctions":16,"virtualF
unctions":[]}]}}
{"level":"Level(-
4)","ts":1616794345.5763638,"logger":"daemon.NodeConfigurator.applyConfig","msg":"co
nfiguring PF","requestedConfig":{"pciAddress":"
0000:af:00.0","pfDriver":"pci-pf-stub","vfDriver":"vfio-pci","vfAmount":2,"bbDevConfig":
{"acc100":{"pfMode":false,"numVfBundles":16,"maxQueueSize":1
024,"uplink4G":
{"numQueueGroups":4,"numAqsPerGroups":16,"aqDepthLog2":4},"downlink4G":
{"numQueueGroups":4,"numAqsPerGroups":16,"aqDepthLog2":4},"uplink5G":
{"numQueueGroups":0,"numAqsPerGroups":16,"aqDepthLog2":4},"downlink5G":
{"numQueueGroups":0,"numAqsPerGroups":16,"aqDepthLog2":4}}}}}
{"level":"Level(-
4)","ts":1616794345.5774765,"logger":"daemon.NodeConfigurator.loadModule","msg":"ex
ecuting command","cmd":"/usr/sbin/chroot /host/ modprobe pci-pf-stub"}
{"level":"Level(-
4)","ts":1616794345.5842702,"logger":"daemon.NodeConfigurator.loadModule","msg":"co
mmands output","output":""}
{"level":"Level(-
4)","ts":1616794345.5843055,"logger":"daemon.NodeConfigurator.loadModule","msg":"ex
ecuting command","cmd":"/usr/sbin/chroot /host/ modprobe vfio-pci"}
{"level":"Level(-
4)","ts":1616794345.6090655,"logger":"daemon.NodeConfigurator.loadModule","msg":"co
mmands output","output":""}
{"level":"Level(-
2)","ts":1616794345.6091156,"logger":"daemon.NodeConfigurator","msg":"device's
driver_override path","path":"/sys/bus/pci/devices/0000:af:00.0/driver_override"}
{"level":"Level(-
2)","ts":1616794345.6091807,"logger":"daemon.NodeConfigurator","msg":"driver bind
path","path":"/sys/bus/pci/drivers/pci-pf-stub/bind"}
{"level":"Level(-
2)","ts":1616794345.7488534,"logger":"daemon.NodeConfigurator","msg":"device's
driver_override path","path":"/sys/bus/pci/devices/0000:b0:00.0/driver_override"}
{"level":"Level(-
2)","ts":1616794345.748938,"logger":"daemon.NodeConfigurator","msg":"driver bind
path","path":"/sys/bus/pci/drivers/vfio-pci/bind"}
{"level":"Level(-
2)","ts":1616794345.7492096,"logger":"daemon.NodeConfigurator","msg":"device's
driver_override path","path":"/sys/bus/pci/devices/0000:b0:00.1/driver_override"}
{"level":"Level(-
2)","ts":1616794345.7492566,"logger":"daemon.NodeConfigurator","msg":"driver bind
path","path":"/sys/bus/pci/drivers/vfio-pci/bind"}
```

```
{"level":"Level(-
4)","ts":1616794345.74968,"logger":"daemon.NodeConfigurator.applyConfig","msg":"exec
uting command","cmd":"/sriov_workdir/pf_bb_config ACC100 -c
/sriov_artifacts/0000:af:00.0.ini -p 0000:af:00.0"}
{"level":"Level(-
4)","ts":1616794346.5203931,"logger":"daemon.NodeConfigurator.applyConfig","msg":"co
mmands output","output":"Queue Groups: 0 5GUL, 0 5GDL, 4 4GUL, 4 4GDL\nNumber
of 5GUL engines 8\nConfiguration in VF mode\nPF ACC100 configuration
complete\nACC100 PF [0000:af:00.0] configuration complete!\n\n"}
{"level":"Level(-
4)","ts":1616794346.520459,"logger":"daemon.NodeConfigurator.enableMasterBus","msg"
:"executing command","cmd":"/usr/sbin/chroot /host/ setpci -v -s 0000:af:00.0
COMMAND"}
{"level":"Level(-
4)","ts":1616794346.5458736,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"commands output","output":"0000:af:00.0 @04 = 0142\n"}
{"level":"Level(-
4)","ts":1616794346.5459251,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"executing command","cmd":"/usr/sbin/chroot /host/ setpci -v -s 0000:af:00.0
COMMAND=0146"}
{"level":"Level(-
4)","ts":1616794346.5795262,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"commands output","output":"0000:af:00.0 @04 0146\n"}
{"level":"Level(-
2)","ts":1616794346.5795407,"logger":"daemon.NodeConfigurator.enableMasterBus","ms
g":"MasterBus set","pci":"0000:af:00.0","output":"0000:af:00.0 @04 0146\n"}
{"level":"Level(-
4)","ts":1616794346.6867144,"logger":"daemon.drainhelper.Run()","msg":"worker
function - end","performUncordon":true}
{"level":"Level(-
4)","ts":1616794346.6867719,"logger":"daemon.drainhelper.Run()","msg":"uncordoning
node"}
{"level":"Level(-
4)","ts":1616794346.6896322,"logger":"daemon.drainhelper.uncordon()","msg":"starting
uncordon attempts"}
{"level":"Level(-
2)","ts":1616794346.69735,"logger":"daemon.drainhelper.uncordon()","msg":"node
uncordoned"}
{"level":"Level(-
4)","ts":1616794346.6973662,"logger":"daemon.drainhelper.Run()","msg":"cancelling the
context to finish the leadership"}
{"level":"Level(-
4)","ts":1616794346.7029872,"logger":"daemon.drainhelper.Run()","msg":"stopped
leading"}
{"level":"Level(-
4)","ts":1616794346.7030034,"logger":"daemon.drainhelper","msg":"releasing the lock
(bug mitigation)"}
{"level":"Level(-
4)","ts":1616794346.8040674,"logger":"daemon.updateInventory","msg":"obtained
inventory","inv":{"sriovAccelerators":
[{"vendorID":"8086","deviceID":"0b32","pciAddress":"0000:20:00.0","driver":"","maxVirtualF
unctions":1,"virtualFunctions":[]},
{"vendorID":"8086","deviceID":"0d5c","pciAddress":"0000:af:00.0","driver":"pci-pf-
stub","maxVirtualFunctions":16,"virtualFunctions":
[{"pciAddress":"0000:b0:00.0","driver":"vfio-pci","deviceID":"0d5d"},
{"pciAddress":"0000:b0:00.1","driver":"vfio-pci","deviceID":"0d5d"}]}]}}
```

```
{"level":"Level(-4)","ts":1616794346.9058325,"logger":"daemon","msg":"Update ignored, generation unchanged"}
{"level":"Level(-2)","ts":1616794346.9065044,"logger":"daemon.Reconcile","msg":"Reconciled","namespace":"vran-acceleration-operators","name":"pg-itengdvs02r.altera.com"}
```


3. カードの FEC 設定を確認します。

```
$ oc get sriovfecnodeconfig node1 -o yaml
```

出力例


```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T11:46:22Z"
    message: Configured successfully
    observedGeneration: 1
    reason: Succeeded
    status: "True"
    type: Configured
  inventory:
    sriovAccelerators:
    - deviceID: 0d5c (1)
      driver: pci-pf-stub
      maxVirtualFunctions: 16
      pciAddress: 0000:af:00.0
      vendorID: "8086"
      virtualFunctions:
      - deviceID: 0d5d (2)
        driver: vfio-pci
        pciAddress: 0000:b0:00.0
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.1
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.2
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.3
      - deviceID: 0d5d
        driver: vfio-pci
        pciAddress: 0000:b0:00.4
```


1. **0d5c** の値は、FEC デバイスの **deviceID** の物理機能です。
2. **0d5d** の値は、FEC デバイスの **deviceID** 仮想機能です。

17.2.4. OpenNESS でのアプリケーション Pod アクセスの確認および ACC100 の使用

OpenNESS は、あらゆるタイプのネットワーク上のアプリケーションおよびネットワーク機能をオンボーディングおよび管理するために使用できるエッジコンピューティングソフトウェアツールキットです。

SR-IOV バインディング、デバイスプラグイン、ワイヤレス Base Band Device (bbdev) 設定、root 以外の Pod 内での SR-IOV (FEC) VF 機能など、すべての OpenNESS 機能がすべて連携していることを確認するには、イメージをビルドし、デバイスの単純な検証アプリケーションを実行できます。

詳細は、openess.org にアクセスします。

前提条件

- ワイヤレス FEC Accelerator 向け OpenNESS SR-IOV Operator でインストールされる 1 つまたは複数のノード
- Performance Addon Operator で設定されたリアルタイムカーネルおよび Huge Page

手順

1. 以下のアクションを実行して、テスト用の namespace を作成します。
   a. 以下の例のように **test-bbdev-namespace.yaml** という名前のファイルを作成し、**test-bbdev** namespace を定義します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-bbdev
  labels:
    openshift.io/run-level: "1"
```

   b. 以下のコマンドを実行して namespace を作成します。

```
$ oc create -f test-bbdev-namespace.yaml
```

2. 以下の **Pod** 仕様を作成してから、YAML を **pod-test.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-bbdev-sample-app
  namespace: test-bbdev (1)
spec:
  containers:
  - securityContext:
      privileged: false
      capabilities:
        add:
        - IPC_LOCK
        - SYS_NICE
    name: bbdev-sample-app
    image: bbdev-sample-app:1.0 (2)
    command: [ "sudo", "/bin/bash", "-c", "--" ]
    runAsUser: 0 (3)
    resources:
      requests:
        hugepages-1Gi: 4Gi (4)
        memory: 1Gi
```

```
      cpu: "4" (5)
      intel.com/intel_fec_acc100: '1' (6)
    limits:
      memory: 4Gi
      cpu: "4"
      hugepages-1Gi: 4Gi
      intel.com/intel_fec_acc100: '1'
```

1. 手順 1 で作成した **namespace** を指定します。
2. コンパイルされた DPDK を含むテストイメージを定義します。
3. コンテナーが root ユーザーとして内部で実行するようにします。
4. hugepage サイズ **hugepages-1Gi** および Pod に割り当てられる hugepage の量を指定します。hugepages および分離された CPU は、Performance Addon Operator を使用して設定する必要があります。
5. CPU の数を指定します。
6. ACC100 5G FEC 設定のテストは、**intel.com/intel_fec_acc100** でサポートされています。

3. Pod を作成します。

```
$ oc apply -f pod-test.yaml
```

4. Pod が作成されていることを確認します。

```
$ oc get pods -n test-bbdev
```

出力例

```
NAME                     READY   STATUS    RESTARTS   AGE
pod-bbdev-sample-app     1/1     Running   0          80s
```

5. リモートシェルを使用して **pod-bbdev-sample-app** にログインします。

```
$ oc rsh pod-bbdev-sample-app
```

出力例

```
sh-4.4#
```

6. Pod に割り当てられる VF を出力します。

```
sh-4.4# printenv | grep INTEL_FEC
```

出力例

```
PCIDEVICE_INTEL_COM_INTEL_FEC_ACC100=0.0.0.0:1d.00.0 (1)
```

1 これは、仮想関数の PCI アドレスです。

7. **test-bbdev** ディレクトリーに移動します。

```
sh-4.4# cd test/test-bbdev/
```

8. Pod に割り当てられている CPU を確認します。

```
sh-4.4# export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
sh-4.4# echo ${CPU}
```

これにより、**fec.pod** に割り当てられた CPU が出力されます。

出力例

```
24,25,64,65
```

9. **test-bbdev** アプリケーションを実行してデバイスをテストします。

```
sh-4.4# ./test-bbdev.py -e="-l ${CPU} -a
${PCIDEVICE_INTEL_COM_INTEL_FEC_ACC100}" -c validation \ -n 64 -b 32 -l 1 -v
./test_vectors/*"
```

出力例

```
Executing: ../../build/app/dpdk-test-bbdev -l 24-25,64-65 0000:1d.00.0 -- -n 64 -l 1 -c
validation -v ./test_vectors/bbdev_null.data -b 32
EAL: Detected 80 lcore(s)
EAL: Detected 2 NUMA nodes
Option -w, --pci-whitelist is deprecated, use -a, --allow option instead
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'VA'
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL:   using IOMMU type 1 (Type 1)
EAL: Probe PCI driver: intel_fpga_5ngr_fec_vf (8086:d90) device: 0000:1d.00.0 (socket 1)
EAL: No legacy callbacks, legacy socket not created



===========================================================
Starting Test Suite : BBdev Validation Tests
Test vector file = ldpc_dec_v7813.data
Device 0 queue 16 setup failed
Allocated all queues (id=16) at prio0 on dev0
Device 0 queue 32 setup failed
Allocated all queues (id=32) at prio1 on dev0
Device 0 queue 48 setup failed
Allocated all queues (id=48) at prio2 on dev0
Device 0 queue 64 setup failed
Allocated all queues (id=64) at prio3 on dev0
Device 0 queue 64 setup failed
All queues on dev 0 allocated: 64
```

```
+ ------------------------------------------------------ +
== test: validation
dev:0000:b0:00.0, burst size: 1, num ops: 1, op type: RTE_BBDEV_OP_LDPC_DEC
Operation latency:
        avg: 23092 cycles, 10.0838 us
        min: 23092 cycles, 10.0838 us
        max: 23092 cycles, 10.0838 us
TestCase [ 0] : validation_tc passed
 + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ +
 + Test Suite Summary : BBdev Validation Tests
 + Tests Total :        1
 + Tests Skipped :      0
 + Tests Passed :       1 (1)
 + Tests Failed :       0
 + Tests Lasted :       177.67 ms
 + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ +
```

(1) 一部のテストはスキップできますが、ベクターテストに合格していることを確認してください。

17.3. 関連情報

- OpenNESS Operator for Wireless FEC Accelerators