OpenShift Container Platform 4.5

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: 2021-07-13

OpenShift Container Platform 4.5 スケーラビリティーおよびパフォーマンス

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

法律上の通知

概要

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

目次

第1章 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

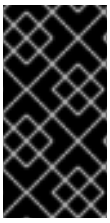
このトピックでは、OpenShift Container Platform のホストについての推奨プラクティスについて説明します。

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。

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) を作成して kubelet パラメーターを編集すること

ができます。

手順

1. 以下を実行します。

```
$ oc get machineconfig
```

これは、選択可能なマシン設定オブジェクトの一覧を提供します。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 の現在の値を確認するには、以下を実行します。

```
# oc describe node <node-ip> | grep Allocatable -A6
```

value: pods: <value> を検索します。

以下は例になります。

```
# oc describe node ip-172-31-128-158.us-east-2.compute.internal | grep Allocatable -A6
```

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。たとえば、**change-maxPods-cr.yaml** を使用します。

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

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: large-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

a. 以下を実行します。

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

b. 以下を実行します。

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

c. 以下を実行します。

```
$ oc get kubeletconfig
```

これにより **set-max-pods**が返されるはずです。

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. ワーカーノードを変更する **maxPods** の有無を確認します。

```
$ oc describe node
```

a. 以下を実行して変更を確認します。

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

これは **True** と **type:Success** のステータスを表示します。

手順

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。

1. 以下を実行します。

```
$ 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。

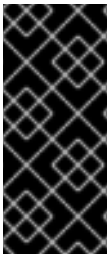
2.3. コントロールプレーンノードのサイジング

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)

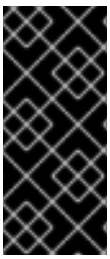
ワーカーノードの数	クラスターの負荷 (namespace)	CPU コア数	メモリー (GB)
25	500	4	16
100	1000	8	32
250	4000	16	96

3 つのマスターノードまたはコントロールプレーンノードを持つクラスターでは、いずれかのノードが停止すると、残りの 2 つのノードが高可用性を維持するために負荷を処理する必要があるために CPU とメモリーの使用量が急上昇します。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。大規模で高密度のクラスターで障害が繰り返し発生しないようにするには、マスターノードでの全体的なリソース使用量を少なくとも利用可能な容量の半分に維持し、使用量の急増に対応できるようにします。マスターノードの 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のインストール方法を使用している場合、実行中の OpenShift Container Platform 4.5 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することはできません。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 4.5 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. 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。データストアの領域を解放するには、デフラグを含む etcd の定期的なメンテナンスを行う必要があります。Prometheus で etcd メトリクスを監視し、etcd がクラスター全体でのアラームを出す前にこのデフラグを実行することを強くお勧めします。いったんアラームが出されると、クラスターはキーの読み取りと削除のみを許可するメンテナンスモードに切り替わります。監視する主要なメトリクスには、現在のクォータ制限である **etcd_server_quota_backend_bytes**、 履歴のコンパクト化後の実際のデータベース使用状況を示す **etcd_mvcc_db_total_size_in_use_in_bytes**、およびデフラグを待機する空き領域を含むデータベースのサイズを示す **etcd_debugging_mvcc_db_total_size_in_bytes** が含まれます。etcd のデフラグについての手順は、「**etcd データのデフラグ**」セクションを参照してください。

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 はディスク上でプロポーザルを永続化させます。他のプロセスからのディスクおよびディスクのアクティビティーにより、fsync のレイテンシーが長くなる可能性があり、etcd がハートビートをミスし、時間通りに新規プロポーザルをディスクにコミットできなくなる可能性があり、要求のタイムアウトや一時的なリーダーが失われる可能性があります。低レイテンシーおよび高スループットの SSD/NVMe ディスクがサポートするマシンで etcd を実行することが強く推奨されます。

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** は etcd ディスク fsync の期間を報告し、**etcd_server_leader_changes_seen_total** はリーダーの変更を報告します。低速なディスクを除外し、ディスクが適度に高速であることを確認するには、**etcd_disk_wal_fsync_duration_seconds_bucket** の 99 パーセンタイルは 10ms 未満である必要があります。

I/O ベンチマークツールの fio は、OpenShift クラスターの作成前後に etcd のハードウェアを検証するために使用できます。fio を実行し、結果を分析します。

podman や docker などのコンテナーランタイムが test 下のマシンにインストールされ、etcd がデータを書き込むパス (/var/lib/etcd) がある場合は、以下を実行します。

手順

podman を使用する場合は、以下を実行します。

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

または、docker を使用している場合は以下を実行します。

```
$ 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 10ms 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します。そのため、そのパフォーマンスはネットワーク入出力 (IO) 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、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]))** は、メンバー間のクライアント要求の複製を終了するまでの etcd のラウンドトリップタイムを報告します。これは 50 ミリ秒未満である必要があります。

2.5. 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 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.us-west-1.compute.internal 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
```

2.6. 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
- ユーザー定義プロジェクトのモニタリング用のコンポーネントを含む、クラスターメトリクスの収集またはモニタリングサービス
- クラスター集計ロギング
- サービスブローカー
- Red Hat Quay
- Red Hat OpenShift Container Storage
- Red Hat Advanced Cluster Manager

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。

2.7. 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。

2.8. 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

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. テキストの以下の行を、オブジェクトの **spec** セクションに追加します。

```
  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。

2.9. 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) 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   v1.18.3
   ```

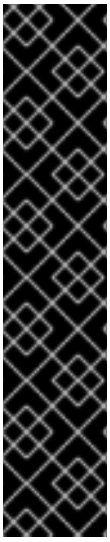
   ロールの一覧に **infra** が含まれているため、Pod は正しいノードで実行されます。

2.10. 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	32
100	8	64
250	32	192

ワーカーノードの数	CPU コア数	メモリー (GB)
500	32	192



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 4.5 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。

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 Two 操作で、これは推奨内容では考慮されません。

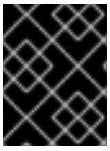
注記

OpenShift Container Platform 4.5 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。

2.11. 追加リソース

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

第3章 クラスタースケーリングに関する推奨プラクティス



重要

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

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

3.1. クラスターのスケーリングに関する推奨プラクティス

クラスターをノード数のより高い値にスケールアップする場合:

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

注記

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

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

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

注記

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

3.2. マシンセットの変更

マシンセットを変更するには、**MachineSet** YAML を編集します。次に、各マシンを削除するか、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します。
- **cluster-admin** パーミッションを持つユーザーとして、**oc** にログインします。

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。

3.3. マシンのヘルスチェック

MachineHealthCheck リソースを使用して、クラスター内のマシンが正常ではないとみなされる条件を定義できます。条件に一致するマシンは自動的に修復されます。

マシンの正常性を監視するには、監視する一連のマシンのラベルや、**NotReady** ステータスの期間を 15 分にしたり、 node-problem-detector に永続的な条件を表示したりするなど、チェックする条件を含む **MachineHealthCheck** カスタムリソース (CR) を作成します。

MachineHealthCheck CR を観察するコントローラーは定義した条件の有無をチェックします。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、新規マシンが代わりに作成されます。マシンが削除されると、**machine deleted** イベントが表示されます。

注記

マスターロールを持つマシンの場合、マシンのヘルスチェックは正常でないノードの数を報告しますが、マシンは削除されません。以下は例になります。

出力例

```
$ oc get machinehealthcheck example -n openshift-machine-api
```

```
NAME      MAXUNHEALTHY   EXPECTEDMACHINES   CURRENTHEALTHY
example   40%            3                  1
```

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、コントローラーはマシンの削除を停止し、手動で介入する必要があります。

チェックを停止するには、カスタムリソースを削除します。

3.3.1. ベアメタル上の MachineHealthCheck

ベアメタルクラスターでのマシンの削除により、ベアメタルホストの再プロビジョニングがトリガーされます。通常、ベアメタルの再プロビジョニングは長いプロセスで、クラスターにコンピュートリソースがなくなり、アプリケーションが中断される可能性があります。デフォルトの修復プロセスをマシンの削除からホストの電源サイクルに切り換えるには、MachineHealthCheck リソースに **machine.openshift.io/remediation-strategy: external-baremetal** アノテーションを付けます。

アノテーションの設定後に、BMC 認証情報を使用して正常でないマシンの電源が入れ直されます。

3.3.2. マシンヘルスチェックのデプロイ時の制限

マシンヘルスチェックをデプロイする前に考慮すべき制限事項があります。

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

3.4. サンプル MACHINEHEALTHCHECK リソース

MachineHealthCheck リソースは以下の YAML ファイルのようになります。

ベアメタルの MachineHealthCheck

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example (1)
  namespace: openshift-machine-api
  annotations:
    machine.openshift.io/remediation-strategy: external-baremetal (2)
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> (3)
      machine.openshift.io/cluster-api-machine-type: <role> (4)
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> (5)
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" (6)
    status: "False"
  - type:    "Ready"
    timeout: "300s" (7)
    status: "Unknown"
  maxUnhealthy: "40%" (8)
  nodeStartupTimeout: "10m" (9)
```

(1) デプロイするマシンヘルスチェックの名前を指定します。

(2) ベアメタルクラスターの場合、電源サイクルの修復を有効にするために **machine.openshift.io/remediation-strategy: external-baremetal** アノテーションを **annotations** セクションに含める必要があります。この修復ストラテジーにより、正常でないホストはクラスターから削除される代わりに、再起動されます。

(3) (4) 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。

(5) 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。

(6) (7) 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。

(8) ターゲットプールで許可される正常でないマシンの量を指定します。これはパーセンテージまたは整数として設定できます。

(9) 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。

他のすべてのインストールタイプの MachineHealthCheck

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

1 デプロイするマシンヘルスチェックの名前を指定します。

2 3 チェックする必要のあるマシンプールのラベルを指定します。

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

5 6 ノードの状態のタイムアウト期間を指定します。タイムアウト期間の条件が満たされると、マシンは修正されます。タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。

7 ターゲットプールで許可される正常でないマシンの量を指定します。これはパーセンテージまたは整数として設定できます。

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

注記

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

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

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

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

重要

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

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

3.4.1.1. 絶対値を使用した maxUnhealthy の設定

maxUnhealthy が **2** に設定される場合:

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

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

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

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

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

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

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

注記

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

3.5. MACHINEHEALTHCHECK リソースの作成

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

前提条件

- **oc** コマンドラインインターフェースをインストールします。

手順

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

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

第4章 NODE TUNING OPERATOR の使用

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

4.1. NODE TUNING OPERATOR について

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

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

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

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

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

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

手順

1. 以下を実行します。

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

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

警告

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

4.3. クラスターに設定されるデフォルトのプロファイル

以下は、クラスターに設定されるデフォルトのプロファイルです。

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  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
```

4.4. 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
```

4.5. カスタムチューニング仕様

Operator のカスタムリソース (CR) には 2 つの重要なセクションがあります。1 つ目のセクションの **profile:** は Tuned プロファイルおよびそれらの名前の一覧です。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。

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 plug-ins 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
```

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**

<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>** をマシン設定プールのノードセレクターに一致するすべてのノードに設定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。

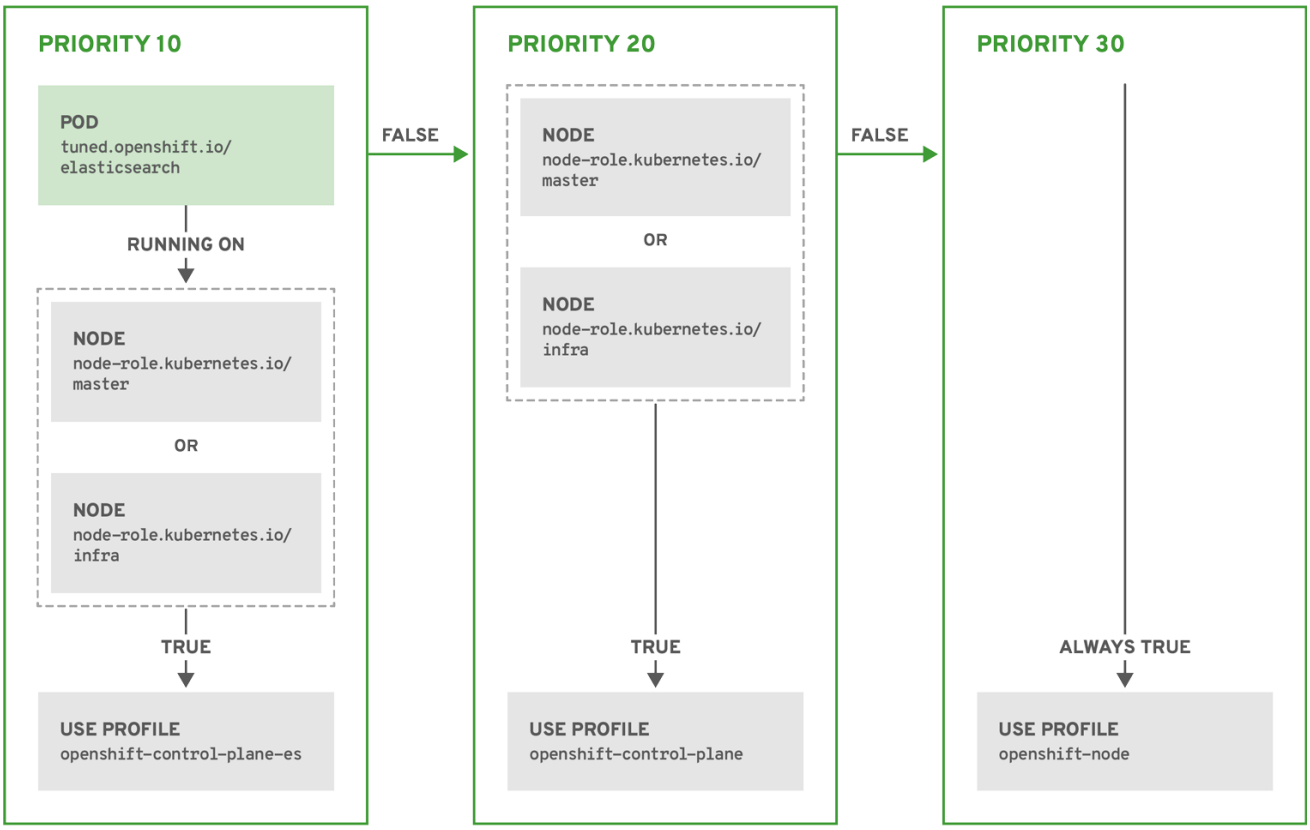
例: ノード/Pod ラベルベースのマッチング

```
- 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
```

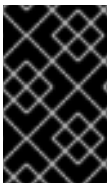
ノードの再起動を最小限にするには、ターゲットノードにマシン設定プールのノードセレクターが一致するラベルを使用してラベルを付け、上記の Tuned CR を作成してから、最後にカスタムのマシン設定プール自体を作成します。

4.6. 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。

4.7. サポートされている TUNED デーモンプラグイン

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

- audio
- cpu
- disk
- eeepc_she
- modules
- mounts
- net
- scheduler

- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm

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5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。

5.1. 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:4.5
```

5.2. 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:4.5 /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:4.5 \
/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。

5.3. 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 namespaces (プロジェクト) を複数作成します。

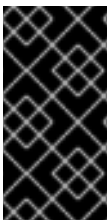
5.3.1. クラスターローダー設定ファイルの例

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。

5.3.2. 設定フィールド

表5.1 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 namespaces (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5.2 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	キーと値のペアー覧。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	キーと値のペアー覧。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5.3 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5.4 **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5.5 **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5.6 **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5.7 **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。

5.4. 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6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

6.1. 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 階層 (quality of service) **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7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、「Using CPU Manager」を参照してください。

7.1. 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。

7.2. TOPOLOGY MANAGER のセットアップ

Topology Manager を使用するには、**LatencySensitive** 機能ゲートを有効にし、**cpumanager-enabled** カスタムリソース (CR) で Topology Manager 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. **FeatureGate** オブジェクトを編集して **LatencySensitive** 機能セットを追加します。

```
$ oc edit featuregate/cluster
```

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: 2020-06-05T14:41:09Z
  generation: 2
  managedFields:
  - apiVersion: config.openshift.io/v1
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .: {}
          f:release.openshift.io/create-only: {}
      f:spec: {}
    manager: cluster-version-operator
    operation: Update
    time: 2020-06-05T14:41:09Z
  - apiVersion: config.openshift.io/v1
    fieldsType: FieldsV1
    fieldsV1:
      f:spec:
        f:featureSet: {}
    manager: oc
    operation: Update
    time: 2020-06-05T15:21:44Z
  name: cluster
  resourceVersion: "28457"
  selfLink: /apis/config.openshift.io/v1/featuregates/cluster
  uid: e802e840-89ee-4137-a7e5-ca15fd2806f8
```

```
spec:
  featureSet: LatencySensitive (1)
...
```

(1) **LatencySensitive** 機能セットをコンマ区切り一覧に追加します。

2. **cpumanager-enabled** カスタムリソース (CR) で Topology Manager 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 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。

追加リソース

CPU マネージャーの詳細は、「Using CPU Manager」を参照してください。

7.3. 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8章 CLUSTER MONITORING OPERATOR のスケーリング

OpenShift Container Platform は、Cluster Monitoring Operator が収集し、Prometheus ベースのモニタリングスタックに保存するメトリクスを公開します。管理者は、Grafana という 1 つのダッシュボードインターフェー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。

8.1. 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8.1 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。

8.2. クラスターモニタリングの設定

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。以下は例になります。

```
apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusOperator:
      baseImage: quay.io/coreos/prometheus-operator
      prometheusConfigReloaderBaseImage: quay.io/coreos/prometheus-config-reloader
      configReloaderBaseImage: quay.io/coreos/configmap-reload
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} 1
      baseImage: openshift/prometheus
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} 2
    alertmanagerMain:
      baseImage: openshift/prometheus-alertmanager
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{ALERTMANAGER_STORAGE_SIZE}} 3
    nodeExporter:
      baseImage: openshift/prometheus-node-exporter
    kubeRbacProxy:
      baseImage: quay.io/coreos/kube-rbac-proxy
    kubeStateMetrics:
      baseImage: quay.io/coreos/kube-state-metrics
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    grafana:
      baseImage: grafana/grafana
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    auth:
      baseImage: openshift/oauth-proxy
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
```

```
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、サフィックス s、m、h、d のいずれかを使用する単位で測定されます。
2. 標準の値は **PROMETHEUS_STORAGE_SIZE=2000Gi** です。ストレージの値には、サフィックス 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。
3. 標準の値は **ALERTMANAGER_STORAGE_SIZE=20Gi** です。ストレージの値には、サフィックス 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. 保持期間とストレージサイズなどの値を設定します。
3. 以下を実行して変更を適用します。

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

第9章 オブジェクトの最大値に合わせた環境計画

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

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

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

9.1. 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。
namespaces 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
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

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 v4 (OCS v4) のみが本書で説明されているノードごとの Pod 数に対応することができました。

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。

9.2. OPENSHIFT CONTAINER PLATFORM のテスト済みのクラスターの最大値

最大値のタイプ	4.1 テスト済みの最大値	4.2 テスト済みの最大値	4.3 テスト済みの最大値	4.4 テスト済みの最大値	4.5 テスト済みの最大値
ノード数	2,000	2,000	2,000	250	500
Pod 数 [1]	150,000	150,000	150,000	62,500	62,500
ノードあたりの Pod 数	250	250	500	500	500
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。	デフォルト値はありません。	デフォルト値はありません。	デフォルト値はありません。
namespaces 数 [2]	10,000	10,000	10,000	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）- 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	10,000（デフォルト Pod RAM 512 Mi）- Source-to-Image (S2I) ビルドストラテジー	10,000（デフォルト Pod RAM 512 Mi）- Source-to-Image (S2I) ビルドストラテジー

最大値のタイプ	4.1 テスト済みの最大値	4.2 テスト済みの最大値	4.3 テスト済みの最大値	4.4 テスト済みの最大値	4.5 テスト済みの最大値
namespace ごとの Pod 数 [3]	25,000	25,000	25,000	25,000	25,000
サービス数 [4]	10,000	10,000	10,000	10,000	10,000
namespace ごとのサービス数	5,000	5,000	5,000	5,000	5,000
サービスごとのバックエンド数	5,000	5,000	5,000	5,000	5,000
namespace ごとのデプロイメント数 [3]	2,000	2,000	2,000	2,000	2,000

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. 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。

3. 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。

4. 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。

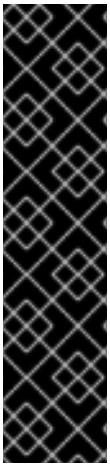
9.3. クラスターの最大値がテスト済みの OPENSHIFT CONTAINER PLATFORM 環境および設定

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	250	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。

9.4. 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
```

ここでは、以下のようになります。

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

9.5. 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ごとのコストなどの要因を考慮する必要があります。

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

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

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

アプリケーション Pod は環境変数または DNS のいずれかを使用してサービスにアクセスできます。環境変数を使用する場合、それぞれのアクティブなサービスについて、変数が Pod がノードで実行される際に kubelet によって挿入されます。クラスター対応の DNS サーバーは、Kubernetes API で新規サービスの有無を監視し、それぞれに DNS レコードのセットを作成します。DNS がクラスター全体で有効にされている場合、すべての Pod は DNS 名でサービスを自動的に解決できるはずです。DNS を使用したサービス検出は、5000 サービスを超える使用できる場合があります。サービス検出に環境変数を使用する場合、引数の一覧は 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。

```
 ---
   Kind: Template
   apiVersion: v1
   metadata:
     name: deploymentConfigTemplate
     creationTimestamp:
     annotations:
       description: This template will create a deploymentConfig with 1 replica, 4 env vars and a
service.
       tags: ''
   objects:
   - kind: DeploymentConfig
     apiVersion: v1
     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${IDENTIFIER}
      triggers:
      - type: ConfigChange
      strategy:
        type: Rolling
  - kind: Service
    apiVersion: v1
    metadata:
      name: service${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: deploymentConfigTemplate
```

第10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。

10.1. 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10.1 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
Block	● 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。
File	● 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 4.5 ではサポートされていません。

10.2. 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10.2 設定可能な推奨ストレージ技術

ストレージタイプ	ROX[1]	RWX[2]	レジストリー	スケーリングされたレジストリー	メトリクス[3]	ロギング	アプリ
Block	Yes[4]	No	設定可能	設定不可	推奨	推奨	推奨
File	Yes[4]	Yes	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	Yes	Yes	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。

10.2.1. 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。

10.2.1.1. 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。

10.2.1.2. 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し、リードアフターライト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。
- 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スケーリングされた/HA の 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。
- 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。

10.2.1.3. 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。

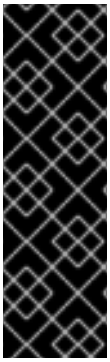
重要

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。

10.2.1.4. ロギング

OpenShift Container 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。
- 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スケーリングされた/HA の 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。
- オブジェクトストレージは設定できません。



重要

テストにより、NFS サーバーを 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、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。

10.2.1.5. 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。

10.2.2. 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

- OpenShift Container Platform Internal **etcd**: **etcd** の信頼性を最も高く保つには、一貫してレイテンシーが最も低くなるストレージ技術が推奨されます。
- NVMe や SSD などのシリアル書き込み (fsync) を迅速に処理するストレージで **etcd** を使用することが強く推奨されます。Ceph、NFS、およびスピニングディスクは推奨されません。

- 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。

10.3. 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10.3 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	備考	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11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。

11.1. 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、「Configuring Ingress Controller sharding by using route labels」および「Configuring Ingress Controller sharding by using namespace labels 」を参照してください。

11.2. 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12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ースカード (NIC) のオフロード、マルチキュー、ethtool の設定を使用してチューニングが可能です。

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。

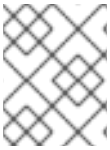
12.1. ネットワークでの MTU の最適化

重要な Maximum Transmission Unit (MTU) が 2 つあります (ネットワークインターフェースカード (NIC) MTU と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。

12.2. 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。

12.3. 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。

追加リソース

- 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13章 HUGE PAGE の機能およびそれらがアプリケーションによって消費される仕組み

13.1. HUGE PAGE の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 256,000 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピングの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

Huge Page とは、4Ki より大きいメモリーページのことです。x86_64 アーキテクチャーでは、2Mi と 1Gi の 2 つが一般的な Huge Page サイズです。別のアーキテクチャーではサイズは異なります。Huge Page を使用するには、アプリケーションが認識できるようにコードを書き込む必要があります。Transparent Huge Pages (THP) は、アプリケーションによる認識なしに、Huge Page の管理を自動化しようとしますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

OpenShift Container Platform では、Pod のアプリケーションが事前に割り当てられた Huge Page を割り当て、消費することができます。

13.2. HUGE PAGE がアプリケーションによって消費される仕組み

ノードは、Huge Page の容量をレポートできるように Huge Page を事前に割り当てる必要があります。ノードは、単一サイズの Huge Page のみを事前に割り当てることができます。

Huge Page は、リソース名の **hugepages-<size>** を使用してコンテナーレベルのリソース要件で消費可能です。この場合、サイズは特定のノードでサポートされる整数値を使用した最もコンパクトなバイナリー表記です。たとえば、ノードが 2048KiB ページサイズをサポートする場合、これはスケジュール可能なリソース **hugepages-2Mi** を公開します。CPU やメモリーとは異なり、Huge Page はオーバーコミットをサポートしません。

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
```

```
    name: hugepage
  resources:
    limits:
      hugepages-2Mi: 100Mi (1)
      memory: "1Gi"
      cpu: "1"
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

(1) **hugepages** のメモリー量は、実際に割り当てる量に指定します。この値は、ページサイズで乗算した **hugepages** のメモリー量に指定しないでください。たとえば、Huge Page サイズが 2MB と仮定し、アプリケーションに Huge Page でバックアップする RAM 100 MB を使用する場合には、Huge Page は 50 に指定します。OpenShift Container Platform により、計算処理が実行されます。上記の例にあるように、**100MB** を直接指定できます。

指定されたサイズの Huge Page の割り当て

プラットフォームによっては、複数の Huge Page サイズをサポートするものもあります。特定のサイズの Huge Page を割り当てるには、Huge Page の起動コマンドパラメーターの前に、Huge Page サイズの選択パラメーター **hugepagesz=<size>** を指定してください。**<size>** の値は、バイトで指定する必要があります。その際、オプションでスケールサフィックス [**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。

追加リソース

- 「Transparent Huge Page の設定」

13.3. 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。

13.3.1. 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。