OpenShift Container Platform 4.1

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 Red Hat OpenShift Container Platform 4.1 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: 2020-05-15

OpenShift Container Platform 4.1 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 Red Hat OpenShift Container Platform 4.1 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、Red Hat OpenShift Container Platform 環境のパフォーマンスを最適化する方法について説明します。

目次

第1章 ホストについての推奨されるプラクティス

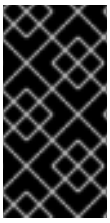
このトピックでは、OpenShift Container Platform のホストについての推奨プラクティスについて説明します。

1.1. ノードホストについての推奨プラクティス

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

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

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



重要

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

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** の値を超えることができません。

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

```
kubeletConfig:
  maxPods: 250
```

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

kubelet 設定は、現時点で Ignition 設定としてシリアル化されているため、直接編集することができます。ただし、新規の kubelet-config-controller も Machine Config Controller (MCC) に追加されます。これにより、KubeletConfig カスタムリソース (CR) を作成して 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
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 数を設定するには、kubelet 設定を含む YAML ファイルを作成します。たとえば、**max-worker-pods.yaml** を使用します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigSelector: 01-worker-kubelet
  kubeletConfig:
    maxPods: 250
```

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 **5** (**kubeAPIQPS** の場合) および **10** (**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。以下を実行します。

+

```
$ oc edit machineconfigpool worker
```

1. **maxUnavailable** を必要な値に設定します。

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

重要

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

1.3. マスターノードのサイジング

マスター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。マスター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。

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	64

重要

実行中の OpenShift Container Platform 4.1 クラスターでマスターノードのサイズを変更することはできないため、ノードの合計数を見積もり、インストール時にマスターの推奨されるサイズを使用する必要があります。

注記

OpenShift Container Platform 4.1 では、デフォルトで CPU コア (500 ミリコア) の半分がシステムによって予約されます (OpenShift Container Platform 3.11 以前のバージョンと比較)。サイズはこれを考慮に入れて決定されます。

1.4. 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。

1.5. 追加リソース

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

第2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。

2.1. 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 (現時点でこれはテクノロジープレビュー機能です)。Operator は、コンテナー化された OpenShift Container Platform の Tuned デーモンを Kubernetes DaemonSet 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。

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。

2.2. 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
```

2.3. 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/v1alpha1
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=>131072
```

```
    net.netfilter.nf_conntrack_max=1048576
    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
    [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
- name: "openshift-control-plane-es"
  data: |
    [main]
    summary=Optimize systems running ES on OpenShift control-plane
    include=openshift-control-plane
    [sysctl]
    vm.max_map_count=262144
- name: "openshift-node-es"
  data: |
    [main]
    summary=Optimize systems running ES on OpenShift nodes
    include=openshift-node
    [sysctl]
    vm.max_map_count=262144
 recommend:
- profile: "openshift-control-plane-es"
  priority: 10
  match:
```

```
    - label: "tuned.openshift.io/elasticsearch"
      type: "pod"
      match:
      - label: "node-role.kubernetes.io/master"
      - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node-es"
    priority: 20
    match:
    - label: "tuned.openshift.io/elasticsearch"
      type: "pod"

  - profile: "openshift-control-plane"
    priority: 30
    match:
    - label: "node-role.kubernetes.io/master"
    - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node"
priority: 40
```

重要

カスタムチューニング仕様のカスタムプロファイルはテクノロジープレビュー機能としてのみ利用可能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

2.4. 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:** はチューニングされた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 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:
- match:                          # optional; if omitted, profile match is assumed unless a profile with a
higher matches first
  <match>                          # an optional array
  priority: <priority>              # profile ordering priority, lower numbers mean higher priority (0 is the
highest priority)
  profile: <tuned_profile_name>       # e.g. tuned_profile_1

# ...

- match:
  <match>
  priority: <priority>
  profile: <tuned_profile_name>       # e.g. tuned_profile_n
```

<match> が省略されている場合は、プロファイルの一致 (例: **true**) があることが想定されます。

<match> は、以下のように再帰的に定義されるオプションの配列です。

```
- label: <label_name>     # node or pod label name
  value: <label_value>    # optional node or pod label value; if omitted, the presence of <label_name>
is enough to match
  type: <label_type>      # optional node or pod type ("node" or "pod"); if omitted, "node" is assumed
  <match>                # an optional <match> array
```

<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>** 配列のいずれかの項目が一致する場合、**<match>** の全体の配列が **true** に評価されます。そのため、配列は論理 OR 演算子として機能します。

例

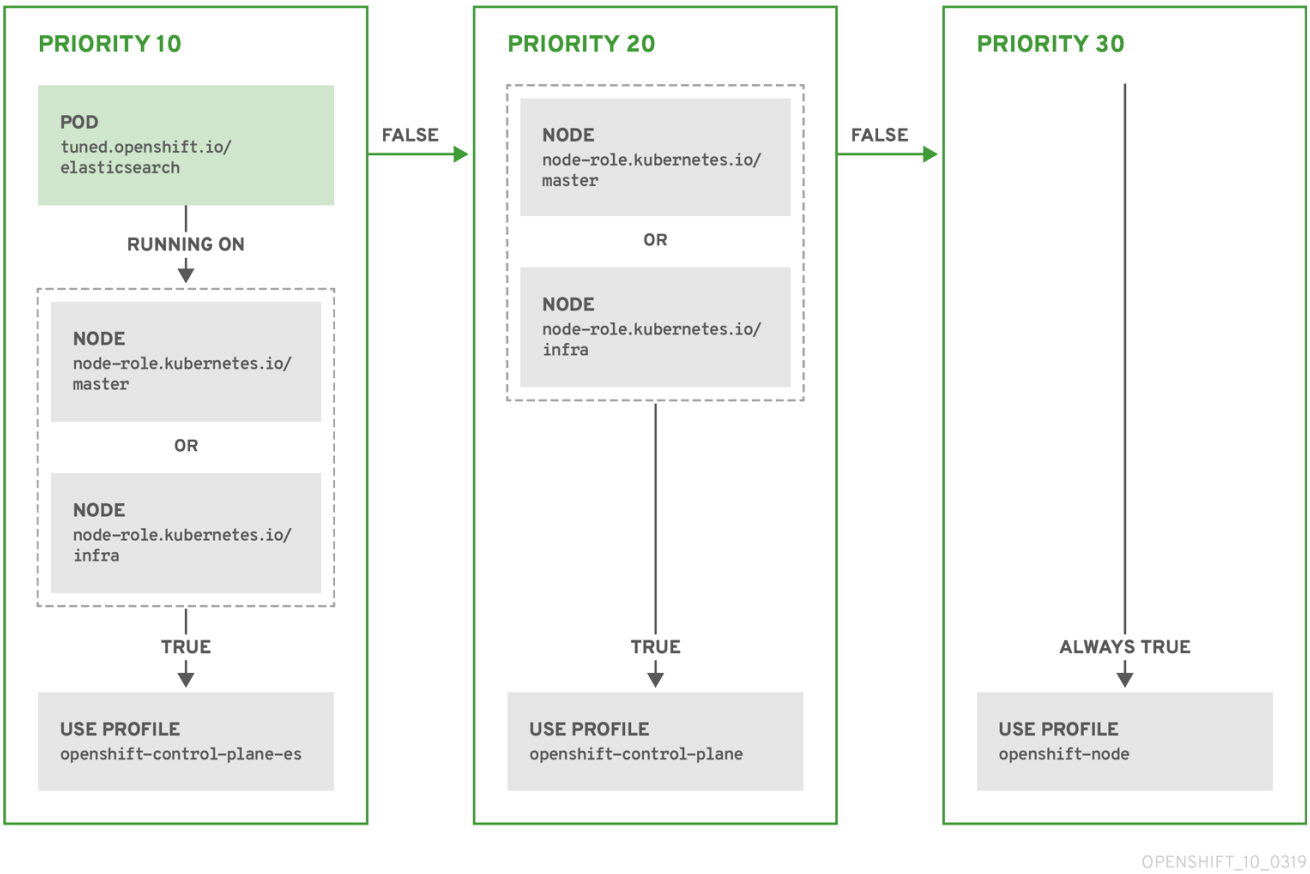
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
```

```
    - 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** ラベルを持つ 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。



2.5. 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、「Available Tuned Plug-ins」および「Getting Started with Tuned」を参照してください。

第3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。

3.1. クラスターローダーのインストール

クラスターローダーは **origin-tests** コンテナーイメージに組み込まれています。

手順

1. **origin-tests** コンテナーイメージをプルするには、以下を実行します。

```
$ sudo podman pull quay.io/openshift/origin-tests:4.1
```

3.2. クラスターローダーの実行

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。

```
$ sudo podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config -i
quay.io/openshift/origin-tests:4.1 /bin/bash -c 'export KUBECONFIG=/root/.kube/config && \
openshift-tests run-test "[Feature:Performance][Serial][Slow] Load cluster should load the \
cluster [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。

```
$ sudo podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config -i
quay.io/openshift/origin-tests:4.1 /bin/bash -c 'export KUBECONFIG=/root/.kube/config && \
export VIPERCONFIG=config/test && \
openshift-tests run-test "[Feature:Performance][Serial][Slow] Load cluster should load the \
cluster [Suite:openshift]"'
```

この例では、**config/** というサブディレクトリーに **test.yml** という設定ファイルが配置されています。コマンドラインで、ファイルタイプと拡張子はツールが自動的に判断するので、設定ファイルを拡張子なしで実行します。

3.3. 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。

クラスターローダーの設定ファイルを **config**/ サブディレクトリーで確認します。これらの設定例で参照される Pod ファイルおよびテンプレートファイルは、**content**/ サブディレクトリーにあります。

3.3.1. 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: ./examples/quickstarts/cakephp-mysql.json

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

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

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

    - num: 1
      basename: clusterloader-rails-postgresql
      tuning: default
      templates:
        - num: 1
          file: ./examples/quickstarts/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

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 podspec 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。

3.3.2. 設定フィールド

表3.1 クラスターローダーの最上位のフィールド

フィールド	Description
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3.2 projects の下にあるフィールド

フィールド	Description
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。

フィールド	Description
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 ConfigMap の名前で、値はこの ConfigMap 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3.3 pods および templates のフィールド

フィールド	Description
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 PodSpec またはテンプレートのいずれかです。
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3.4 tuningsets の下にあるフィールド

フィールド	Description

フィールド	Description
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3.5 tuningsets pods または tuningsets templates の下にあるフィールド

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

フィールド	Description
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3.7 sync の下にあるフィールド

フィールド	Description

フィールド	Description
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。

3.4. 既知の問題

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4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

4.1. 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 **MachineConfigPool** に追加します。

```
metadata:
  creationTimestamp: 2019-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 **KubeletConfig** 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
     cpuManagerReconcilePeriod: 5s
```

5. 動的な **KubeletConfig** 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 **KubeletConfig** 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 **KubeletConfig** を確認します。

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

      "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-4.4# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
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. 1つまたは複数のコアを要求する 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 専用のコアの数になります。

```
# cat cpumanager-pod.yaml
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
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
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
 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
...
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。**kube-reserved** は 500 ミリコアに設定して、**Node Allocatable** の数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5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。OpenShift Container Platform 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。

5.1. PROMETHEUS データベースのストレージ要件

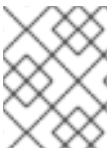
Red Hat では、異なるスケールサイズに応じて各種のテストが実行されました。

表5.1 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。



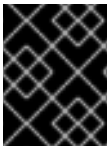
注記

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。

ラボ環境

以前のリリースでは、すべての実験は OpenStack 環境の OpenShift Container Platform で実行されました。

- インフラストラクチャーノード (VM) - 40 コア、157 GB RAM。
- CNS ノード (VM) - 16 コア、62 GB RAM、NVMe ドライブ。



重要

現時点で、OpenStack 環境は OpenShift Container Platform 4.1 用にはサポートされていません。

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。

5.2. 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6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。

6.1. 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、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 [a]	150,000	150,000
ノードあたりの Pod 数	250	500 [b]
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 [c]	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 [d]	25,000	25,000
サービス数 [e]	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 [d]	2,000	2,000

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

[a] ここで表示される Pod 数はテスト Pod の数です。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。

[b] 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** を取得するには、クラスターはカスタム KubeletConfig を使用して **install-config.yaml** ファイルで **hostPrefix** が **22** に指定され、**maxPods** が **500** に設定された状態で作成される必要があります。Persistant Volume Claim（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。

[c] 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。

[d] 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。

[e] 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。

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

制限の種類	3.9 テスト済みの最大値	3.10 テスト済みの最大値	3.11 テスト済みの最大値	4.1 テスト済みの最大値
ノード数	2,000	2,000	2,000	2,000
Pod 数 [a]	150,000	150,000	150,000	150,000
ノードあたりの Pod 数	250	250	250	250
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。
namespace 数 [b]	10,000	10,000	10,000	10,000
ビルド数	10,000 (デフォルトの Pod RAM: 512 Mi)	10,000 (デフォルトの Pod RAM: 512 Mi)	10,000 (デフォルトの Pod RAM: 512 Mi)	10,000 (デフォルトの Pod RAM: 512 Mi)
namespace ごとの Pod 数 [c]	3,000	3,000	25,000	25,000

制限の種類	3.9 テスト済みの最大値	3.10 テスト済みの最大値	3.11 テスト済みの最大値	4.1 テスト済みの最大値
サービス数 [d]	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
サービスごとのバックエンド数	5,000	5,000	5,000	5,000
namespace ごとのデプロイメント数 [c]	2,000	2,000	2,000	2,000

[a] ここで表示される Pod 数はテスト Pod の数です。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。

[b] 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。

[c] 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。

[d] 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。

OpenShift Container Platform 4.1 では、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)。

OpenShift Container Platform 4.1 では、テスト済みのノード制限は、スケーリングテストが高いノード数で実行できるようになるまで低く設定されます。

6.3. 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)/IOPS	カウント	リージョン
マスター/Etcd [a]	r5.4xlarge	16	128	io1	250	3	us-west-2

ノード	フレーバー	vCPU	RAM(GiB)	ディスクタイプ	ディスクサイズ(GiB)/IOPS	カウント	リージョン
インフラ[b]	m5.12xlarge	48	192	gp2	100	3	us-west-2
ワークロード[c]	m5.4xlarge	16	64	gp2	500[d]	1	us-west-2
ワーカー	m5.2xlarge	8	32	gp2	100	3/25/250[e]	us-west-2

[a] 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。

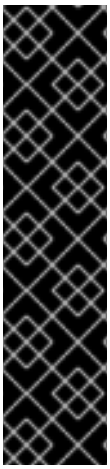
[b] インフラストラクチャーノードは、モニタリング、Ingress およびレジストリーコンポーネントをホストするために使用され、これにより、それらが大規模に実行する場合に必要とするリソースを十分に確保することができます。

[c] 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。

[d] 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。

[e] 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。

6.4. 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 250 の Pod があることを前提として、最低でも 9 つのノードが必要になります。

```
2200 / 250 = 8.8
```

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
```

6.5. 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 (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。

第7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。

7.1. 利用可能な永続ストレージオプション

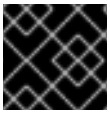
永続ストレージオプションについて理解し、OpenShift Container Platform 環境を最適化できるようにします。

表7.1 利用可能なストレージオプション

ストレージタイプ	Description	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) のネイティブ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 [a]および Vendor NFS
Object	● 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

[a] 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。

ストレージタイプ	Description	例



重要

現時点で、CNS は OpenShift Container Platform 4.1 ではサポートされていません。

7.2. 設定可能な推奨ストレージ技術

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

表7.2 設定可能な推奨ストレージ技術

ストレージタイプ	ROX [a]	RWX [b]	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 [c]	ロギング	アプリ
Block	Yes [d]	No	設定可能	設定不可	推奨	推奨	推奨
File	Yes [d]	Yes	設定可能	設定可能	設定可能 [e]	設定可能 [f]	推奨
Object	Yes	Yes	推奨	推奨	設定不可	設定不可	設定不可 [g]

[a] ReadOnlyMany

[b] ReadWriteMany

[c] 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。

[d] 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。

[e] 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 PersistentVolumeClaims で RWX アクセスモードを設定しないでください。

[f] 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。

[g] オブジェクトストレージは、OpenShift Container Platform の PV/永続ボリューム要求 (PVC: Persistent Volume Claim) 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、3 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。

7.2.1. 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 RHEL でコンテナーイメージレジストリー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 Cassandra、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 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 コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

7.2.1.1. 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:

- 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。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。すべての NAS 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。
- **hostPath** ボリュームは、スケーリングなし/非 HA の OpenShift Container Platform レジストリー用に設定可能ですが、クラスターデプロイメントには推奨しません。

7.2.1.2. 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:

- 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。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。
- 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、ファイルストレージやブロックストレージは推奨しません。
- すべての NAS 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。

7.2.1.3. 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。

重要

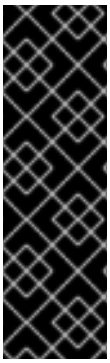
テストの結果、ファイルストレージを使用すると修復不能な大規模な破損が発生することが確認されたため、ファイルストレージをメトリクスで使用することは推奨されません。

これらの問題が検出されない可能性のあるファイルストレージの実装が市場で利用できる可能性があります。OpenShift 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、個別のストレージベンダーにお問い合わせください。

7.2.1.4. 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。
- 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 NAS ストレージを使用することは推奨されません。



重要

テストにより、NFS サーバーを RHEL でコンテナーイメージレジストリー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 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 コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

7.2.1.5. アプリケーション

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

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

7.2.2. 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。
- OpenStack Cinder: OpenStack 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傾向にあります。

第8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。

8.1. ベースラインのルーターパフォーマンス

OpenShift Container Platform 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)

個別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ます。 ルート 100 個を処理し、1kB 静的ページに対応するバックエンドで終端される 100 ルートを処理する単一の HAProxy ルーターは、1 秒ごと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ルート設定が使用され、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。ルータ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。

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。

8.2. ルーター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**) を設定してルータ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。

ルータ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9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

9.1. 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。

9.2. 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。サイズは、特定のノードでサポートされる最もコンパクトなバイナリー表示 (整数値を使用) に置き換えます。たとえば、ノードが 2048KiB のページサイズをサポートする場合は、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 requirements

- 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

- Configuring Transparent Huge Pages

9.3. 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。Node Tuning Operator を使用し、特定のノードで Huge Page を割り当てます。

手順

1. ノードにタグを付け、割り当てる必要のある Huge Page を記述した Tuned プロファイルを適用するノードを Node Tuning Operator が識別できるようにします。

```
$ oc label node <node_using_hugepages> hugepages=true
```

2. 以下の内容でファイルを作成し、これに **hugepages_tuning.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=Configuration for hugepages
      include=openshift-node

      [vm]
      transparent_hugepages=never

      [sysctl]
      vm.nr_hugepages=1024
    name: node-hugepages
  recommend:
  - match: (3)
    - label: hugepages
    priority: 30
    profile: node-hugepages
```

(1) **name** パラメーターの値を **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) プロファイルを **hugepages** ラベルのあるノードに関連付けるには、 **match** セクションを設定します。

3. **hugepages_tuning.yaml** ファイルを使用して、カスタム **hugepages** Tuned プロファイルを作成します。

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

4. プロファイルの作成後、Operator は新規プロファイルを正確なノードに適用し、Huge Page を割り当てます。Huge Page を使用してノードでチューニングされた Pod のログをチェックして、以下を確認します。

```
$ oc logs <tuned_pod_on_node_using_hugepages> \
    -n openshift-cluster-node-tuning-operator | grep 'applied$' | tail -n1

2019-08-08 07:20:41,286 INFO     tuned.daemon.daemon: static tuning from profile 'node-
hugepages' applied
```