



# OpenShift Container Platform 4.5

## インストール後の設定

OpenShift Container Platform の Day 2 オペレーション



# OpenShift Container Platform 4.5 インストール後の設定

---

## OpenShift Container Platform の Day 2 オペレーション

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

## 法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Post-installation\_configuration.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本書では、OpenShift Container Platform のインストール後のアクティビティーについての手順およびガイダンスについて説明します。

## 目次

<b>第1章 インストール後のクラスタータスク</b>	<b>6</b>
1.1. ワーカーノードの調整	6
1.1.1. マシンセットとマシン設定プールの相違点について	6
1.1.2. マシンセットの手動によるスケーリング	6
1.1.3. マシンセットの削除ポリシー	7
1.1.4. クラスタースコープのデフォルトノードセクターの作成	7
1.2. 実稼働環境用のインフラストラクチャーマシンセットの作成	10
1.2.1. マシンセットの作成	10
1.2.2. 専用インフラストラクチャーノードの作成	11
1.2.3. インフラストラクチャーマシンのマシン設定プール作成	12
1.3. マシンセットリソースのインフラストラクチャーノードへの割り当て	15
1.3.1. テイントおよび容認を使用したインフラストラクチャーノードのワークロードのバインディング	16
1.4. リソースのインフラストラクチャーマシンセットへの移行	17
1.4.1. ルーターの移動	17
1.4.2. デフォルトレジストリーの移行	19
1.4.3. モニターリングソリューションの移動	20
1.4.4. クラスターロギングリソースの移動	21
1.5. CLUSTER AUTOSCALER について	25
1.5.1. ClusterAutoscaler リソース定義	26
1.5.2. Cluster Autoscaler のデプロイ	28
1.6. MACHINE AUTOSCALER	28
1.6.1. MachineAutoscaler リソース定義	29
1.6.2. Machine Autoscaler のデプロイ	29
1.7. FEATUREGATE の使用によるテクノロジープレビュー機能の有効化	30
1.8. ETCD タスク	30
1.8.1. etcd 暗号化について	30
1.8.2. etcd 暗号化の有効化	30
1.8.3. etcd 暗号化の無効化	31
1.8.4. etcd データのバックアップ	32
1.8.5. etcd データのデフラグ	34
1.8.6. クラスターの直前の状態への復元	37
1.9. POD の DISRUPTION BUDGET (停止状態の予算)	42
1.9.1. Pod の Disruption Budget (停止状態の予算) を使って起動している Pod の数を指定する方法	42
1.9.2. Pod の Disruption Budget を使って起動している Pod 数の指定	43
1.10. クラウドプロバイダーの認証情報のローテーションまたは削除	44
1.10.1. クラウドプロバイダーの認証情報の削除	44
1.11. 非接続クラスターのイメージストリームの設定	45
1.11.1. サポートデータを収集するためのクラスターの準備	45
関連情報	46
<b>第2章 インストール後のノードタスク</b>	<b>47</b>
2.1. RHEL コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加	47
2.1.1. RHEL コンピュータノードのクラスターへの追加について	47
2.1.2. RHEL コンピュータノードのシステム要件	47
2.1.2.1. 証明書署名要求の管理	48
2.1.3. Playbook 実行のためのマシンの準備	48
2.1.4. RHEL コンピュータノードの準備	50
2.1.5. RHEL コンピュータマシンのクラスターへの追加	51
2.1.6. Ansible ホストファイルの必須パラメーター	52
2.1.7. オプション: RHCOS コンピュータマシンのクラスターからの削除	52
2.2. RHCOS コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加	53

2.2.1. 前提条件	53
2.2.2. ISO イメージを使用した追加の RHCOS マシンの作成	53
2.2.3. PXE または iPXE ブートによる追加の RHCOS マシンの作成	54
2.2.4. マシンの証明書署名要求の承認	56
2.3. マシンヘルスチェックのデプロイ	58
2.3.1. マシンのヘルスチェック	58
2.3.1.1. ベアメタル上の MachineHealthCheck	59
2.3.1.2. マシンヘルスチェックのデプロイ時の制限	59
2.3.2. サンプル MachineHealthCheck リソース	60
2.3.2.1. マシンヘルスチェックによる修復の一時停止 (short-circuiting)	61
2.3.2.1.1. 絶対値を使用した maxUnhealthy の設定	62
2.3.2.1.2. パーセンテージを使用した maxUnhealthy の設定	62
2.3.3. MachineHealthCheck リソースの作成	62
2.3.4. マシンセットの手動によるスケーリング	63
2.3.5. マシンセットとマシン設定プールの相違点について	63
2.4. ノードホストについての推奨プラクティス	64
2.4.1. kubelet パラメーターを編集するための KubeletConfig CRD の作成	64
2.4.2. コントロールプレーンノードのサイジング	67
2.4.3. CPU マネージャーの設定	68
2.5. HUGE PAGE	72
2.5.1. Huge Page の機能	72
2.5.2. Huge Page がアプリケーションによって消費される仕組み	72
2.5.3. Huge Page の設定	74
2.5.3.1. ブート時	74
2.6. デバイスプラグインについて	75
デバイスプラグインの例	76
2.6.1. デバイスプラグインのデプロイ方法	76
2.6.2. デバイスマネージャーについて	77
2.6.3. デバイスマネージャーの有効化	77
2.7. テイントおよび容認 (TOLERATION)	79
2.7.1. テイントおよび容認 (Toleration) について	79
2.7.1.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用方法	81
2.7.1.2. 複数のテイントの使用法	82
2.7.1.3. Pod のスケジューリングとノードの状態 (Taint Nodes By Condition) について	83
2.7.1.4. Pod の状態別エビクションについて (Taint-Based Eviction)	83
2.7.1.5. すべてのテイントの許容	85
2.7.2. テイントおよび容認 (Toleration) の追加	85
2.7.3. マシンセットを使用したテイントおよび容認の追加	86
2.7.4. テイントおよび容認 (Toleration) 使ってユーザーをノードにバインドする	88
2.7.5. テイントおよび容認 (Toleration) を使って特殊ハードウェアを持つノードを制御する	88
2.7.6. テイントおよび容認 (Toleration) の削除	89
2.8. TOPOLOGY MANAGER	90
2.8.1. Topology Manager ポリシー	90
2.8.2. Topology Manager のセットアップ	91
2.8.3. Pod の Topology Manager ポリシーとの対話	92
2.9. リソース要求とオーバーコミット	93
2.10. CLUSTER RESOURCE OVERRIDE OPERATOR を使用したクラスターレベルのオーバーコミット	93
2.10.1. Web コンソールを使用した Cluster Resource Override Operator のインストール	95
2.10.2. CLI を使用した Cluster Resource Override Operator のインストール	97
2.10.3. クラスターレベルのオーバーコミットの設定	100
2.11. ノードレベルのオーバーコミット	101
2.11.1. コンピュートリソースとコンテナについて	101
2.11.1.1. コンテナの CPU 要求について	101

2.11.1.2. コンテナのメモリー要求について	101
2.11.2. オーバーコミットメントと QoS (Quality of Service) クラスについて	101
2.11.2.1. Quality of Service (QoS) 層でのメモリーの予約方法について	102
2.11.3. swap メモリーと QOS について	103
2.11.4. ノードのオーバーコミットについて	103
2.11.5. CPU CFS クォータの使用による CPU 制限の無効化または実行	104
2.11.6. システムリソースのリソース予約	105
2.11.7. ノードのオーバーコミットの無効化	105
2.12. プロジェクトレベルの制限	106
2.12.1. プロジェクトでのオーバーコミットメントの無効化	106
2.13. ガベージコレクションを使用しているノードリソースの解放	106
2.13.1. 終了したコンテナがガベージコレクションによって削除される仕組みについて	106
2.13.2. ガベージコレクションによってイメージが削除される仕組みについて	107
2.13.3. コンテナおよびイメージのガベージコレクションの設定	107
2.14. NODE TUNING OPERATOR の使用	110
2.14.1. Node Tuning Operator 仕様サンプルへのアクセス	110
2.14.2. カスタムチューニング仕様	111
2.14.3. クラスターに設定されるデフォルトのプロファイル	115
2.14.4. サポートされている Tuned デーモンプラグイン	116
2.15. ノードあたりの POD の最大数の設定	117
<b>第3章 インストール後のネットワーク設定</b>	<b>120</b>
3.1. OPENSIFT SDN を使用したネットワークポリシーの設定	120
3.1.1. ネットワークポリシーについて	120
3.1.2. サンプル NetworkPolicy オブジェクト	123
3.1.3. ネットワークポリシーの作成	123
3.1.4. ネットワークポリシーの削除	125
3.1.5. ネットワークポリシーの表示	125
3.1.6. ネットワークポリシーを使用したマルチテナント分離の設定	126
3.1.7. 新規プロジェクトのデフォルトネットワークポリシーの作成	129
3.1.8. 新規プロジェクトのテンプレートの変更	129
3.1.8.1. 新規プロジェクトへのネットワークポリシーの追加	130
3.2. DNS をプライベートに設定する	131
3.3. クラスター全体のプロキシの有効化	133
3.4. CLUSTER NETWORK OPERATOR (CNO) の設定	135
3.5. INGRESS クラスタラフィックの設定	135
3.6. RED HAT OPENSIFT SERVICE MESH でサポートされている設定	135
3.6.1. Red Hat OpenShift Service Mesh でサポートされている Kiali の設定	136
3.6.2. サポートされている Mixer アダプター	136
3.6.3. Red Hat OpenShift Service Mesh のインストールアクティビティー	136
3.7. ルーティングの最適化	137
3.7.1. 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	137
3.7.2. Ingress コントローラー (ルーター) のパフォーマンスの最適化	138
<b>第4章 インストール後のストレージ設定</b>	<b>139</b>
4.1. 動的プロビジョニング	139
4.1.1. 動的プロビジョニングについて	139
4.1.2. 利用可能な動的プロビジョニングプラグイン	139
4.2. ストレージクラスの定義	140
4.2.1. 基本 StorageClass オブジェクト定義	140
4.2.2. ストレージクラスのアノテーション	141
4.2.3. RHOSP Cinder オブジェクトの定義	142
4.2.4. AWS Elastic Block Store (EBS) オブジェクト定義	142

4.2.5. Azure Disk オブジェクト定義	143
4.2.6. Azure File のオブジェクト定義	144
4.2.6.1. Azure File を使用する場合の考慮事項	145
4.2.7. GCE PersistentDisk (gcePD) オブジェクトの定義	146
4.2.8. VMWare vSphere オブジェクトの定義	146
4.3. デフォルトストレージクラスの変更	147
4.4. ストレージの最適化	147
4.5. 利用可能な永続ストレージオプション	148
4.6. 設定可能な推奨のストレージ技術	149
4.6.1. 特定アプリケーションのストレージの推奨事項	149
4.6.1.1. レジストリー	150
4.6.1.2. スケーリングされたレジストリー	150
4.6.1.3. メトリクス	150
4.6.1.4. ロギング	151
4.6.1.5. アプリケーション	151
4.6.2. 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	151
4.7. RED HAT OPENSIFT CONTAINER STORAGE のデプロイ	152
<b>第5章 ユーザー向けの準備</b>	<b>154</b>
5.1. アイデンティティプロバイダー設定について	154
5.1.1. OpenShift Container Platform のアイデンティティプロバイダーについて	154
5.1.2. サポートされるアイデンティティプロバイダー	154
5.1.3. アイデンティティプロバイダーパラメーター	155
5.1.4. アイデンティティプロバイダー CR のサンプル	156
5.2. RBAC の使用によるパーミッションの定義および適用	157
5.2.1. RBAC の概要	157
5.2.1.1. デフォルトのクラスターロール	158
5.2.1.2. 認可の評価	160
5.2.1.2.1. クラスターロールの集計	161
5.2.2. プロジェクトおよび namespace	161
5.2.3. デフォルトプロジェクト	162
5.2.4. クラスターロールおよびバインディングの表示	162
5.2.5. ローカルのロールバインディングの表示	169
5.2.6. ロールのユーザーへの追加	171
5.2.7. ローカルロールの作成	173
5.2.8. クラスターロールの作成	173
5.2.9. ローカルロールバインディングのコマンド	174
5.2.10. クラスターのロールバインディングコマンド	174
5.2.11. クラスター管理者の作成	175
5.3. KUBEADMIN ユーザー	175
5.3.1. kubeadmin ユーザーの削除	175
5.4. イメージ設定リソース	176
5.4.1. イメージコントローラー設定パラメーター	176
5.4.2. イメージ設定内容の設定	178
5.4.2.1. イメージレジストリーアクセス用の追加のトラストストアの設定	179
5.4.2.2. 非セキュアなレジストリー	180
5.4.2.3. イメージレジストリーのリポジトリミラーリングの設定	182
5.5. OPERATORHUB を使用した OPERATOR のインストール	186
5.5.1. Web コンソールを使用した OperatorHub からのインストール	186
5.5.2. CLI を使用した OperatorHub からのインストール	188





## 第1章 インストール後のクラスタータスク

OpenShift Container Platform のインストール後に、クラスターをさらに拡張し、要件に合わせてカスタマイズできます。

### 1.1. ワーカーノードの調整

デプロイメント時にワーカーノードのサイズを誤って設定した場合には、1つ以上の新規マシンセットを作成してそれらをスケールアップしてから、元のマシンセットを削除する前にスケールダウンしてこれらのワーカーノードを調整します。

#### 1.1.1. マシンセットとマシン設定プールの相違点について

**MachineSet** オブジェクトは、クラウドまたはマシンプロバイダーに関する OpenShift Container Platform ノードを記述します。

**MachineConfigPool** オブジェクトにより、**MachineConfigController** コンポーネントがアップグレードのコンテキストでマシンのステータスを定義し、提供できるようになります。

**MachineConfigPool** オブジェクトにより、ユーザーはマシン設定プールの OpenShift Container Platform ノードにアップグレードを展開する方法を設定できます。

**NodeSelector** オブジェクトは **MachineSet** オブジェクト への参照に置き換えることができます。

#### 1.1.2. マシンセットの手動によるスケーリング

マシンセットのマシンのインスタンスを追加したり、削除したりする必要がある場合、マシンセットを手動でスケーリングできます。

本書のガイダンスは、完全に自動化された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に関連します。ユーザーによってプロビジョニングされるインフラストラクチャーのカスタマイズされたインストールにはマシンセットがありません。

#### 前提条件

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

#### 手順

1. クラスターにあるマシンセットを表示します。

```
$ oc get machinesets -n openshift-machine-api
```

マシンセットは **<clusterid>-worker-<aws-region-az>** の形式で一覧表示されます。

2. マシンセットをスケーリングします。

```
$ 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
```

■

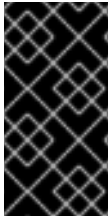
マシンセットをスケールアップまたはスケールダウンできます。新規マシンが利用可能になるまで数分の時間がかかります。

### 1.1.3. マシンセットの削除ポリシー

**Random**、**Newest**、および **Oldest** は3つのサポートされる削除オプションです。デフォルトは **Random** であり、これはマシンセットのスケールダウン時にランダムマシンが選択され、削除されることを意味します。削除ポリシーは、特定のマシンセットを変更し、ユースケースに基づいて設定できます。

```
spec:
  deletePolicy: <delete_policy>
  replicas: <desired_replica_count>
```

削除についての特定のマシンの優先順位は、削除ポリシーにかかわらず、アノテーション **machine.openshift.io/cluster-api-delete-machine** を関係するマシンに追加して設定できます。



#### 重要

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 にスケールリングできません。



#### 注記

カスタムマシンセットは、サービスを特定のノードサービスで実行し、それらのサービスがワーカーのマシンセットのスケールダウン時にコントローラーによって無視されるようにする必要があるユースケースで使用できます。これにより、サービスの中断が回避されます。

### 1.1.4. クラスタスコープのデフォルトノードセクターの作成

クラスター内の作成されたすべての Pod を特定のノードに制限するために、デフォルトのクラスタスコープのノードセクターをノード上のラベルと共に Pod で使用することができます。

クラスタスコープのノードセクターを使用する場合、クラスターで Pod を作成すると、OpenShift Container Platform はデフォルトのノードセクターを Pod に追加し、一致するラベルのあるノードで Pod をスケジュールします。

スケジューラー Operator カスタムリソース (CR) を編集して、クラスタスコープのノードセクターを設定します。ラベルをノード、マシンセット、またはマシン設定に追加します。マシンセットにラベルを追加すると、ノードまたはマシンが停止した場合に、新規ノードにそのラベルが追加されます。ノードまたはマシン設定に追加されるラベルは、ノードまたはマシンが停止すると維持されません。



#### 注記

Pod にキーと値のペアを追加できます。ただし、デフォルトキーの異なる値を追加することはできません。

#### 手順

デフォルトのクラスタスコープのセクターを追加するには、以下を実行します。

1. スケジューラー Operator CR を編集して、デフォルトのクラスタースコープのノードクラスターを追加します。

```
$ oc edit scheduler cluster
```

### ノードセクターを含むスケジューラー Operator CR のサンプル

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...

spec:
  defaultNodeSelector: type=user-node,region=east ❶
  mastersSchedulable: false
  policy:
    name: ""
```

- ❶ 適切な **<key>:<value>** ペアが設定されたノードセクターを追加します。

この変更を加えた後に、**openshift-kube-apiserver** プロジェクトの Pod の再デプロイを待機します。これには数分の時間がかかる場合があります。デフォルトのクラスター全体のノードセクターは、Pod の再起動まで有効になりません。

2. マシンセットを使用するか、またはノードを直接編集してラベルをノードに追加します。

- マシンセットを使用して、ノードの作成時にマシンセットによって管理されるノードにラベルを追加します。

- a. 以下のコマンドを実行してラベルを **MachineSet** オブジェクトに追加します。

```
$ oc patch MachineSet <name> --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
<value>","<key>="<value>"}}] -n openshift-machine-api ❶
```

- ❶ それぞれのラベルに **<key> /<value>** ペアを追加します。

以下に例を示します。

```
$ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
node","region":"east"}}] -n openshift-machine-api
```

- b. **oc edit** コマンドを使用して、ラベルが **MachineSet** オブジェクトに追加されていることを確認します。  
以下に例を示します。

```
$ oc edit MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

### 出力例

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
...
spec:
...
  template:
    metadata:
...
    spec:
      metadata:
        labels:
          region: east
          type: user-node

```

- c. **0** にスケールダウンし、ノードをスケールアップして、そのマシンセットに関連付けられたノードを再デプロイします。  
以下に例を示します。

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

- d. ノードの準備ができ、利用可能な状態になったら、**oc get** コマンドを使用してラベルがノードに追加されていることを確認します。

```
$ oc get nodes -l <key>=<value>
```

以下に例を示します。

```
$ oc get nodes -l type=user-node
```

## 出力例

```

NAME                                STATUS ROLES  AGE  VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp Ready  worker  61s  v1.18.3+002a51f

```

- ラベルをノードに直接追加します。

- a. ノードの **Node** オブジェクトを編集します。

```
$ oc label nodes <name> <key>=<value>
```

たとえば、ノードにラベルを付けるには、以下を実行します。

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node
region=east
```

- b. **oc get** コマンドを使用して、ラベルがノードに追加されていることを確認します。

```
$ oc get nodes -l <key>=<value>,<key>=<value>
```

以下に例を示します。

```
$ oc get nodes -l type=user-node,region=east
```

### 出力例

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49	Ready	worker	17m	v1.18.3+002a51f

## 1.2. 実稼働環境用のインフラストラクチャマシンセットの作成

実稼働デプロイメントでは、インフラストラクチャーコンポーネントを保持するために3つ以上のマシンセットをデプロイします。ロギング集約ソリューションおよびサービスメッシュはどちらも Elasticsearch をデプロインし、Elasticsearch では複数の異なるノードにインストールされる3つのインスタンスが必要です。高可用性を確保するには、これらのノードを複数の異なるアベイラビリティゾーンにインストールし、デプロイします。各アベイラビリティゾーンにそれぞれ異なるマシンセットが必要であるため、3つ以上のマシンセットを作成します。

この手順で使用するのことができるマシンセットの例については、[異なるクラウドのマシンセットの作成](#)を参照してください。

### 1.2.1. マシンセットの作成

インストールプログラムによって作成されるものに加え、独自のマシンセットを作成して、選択する特定のワークロードに対するマシンのコンピュートリソースを動的に管理することができます。

#### 前提条件

- OpenShift Container Platform クラスターをデプロイすること。
- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** パーミッションを持つユーザーとして、**oc** にログインする。

#### 手順

1. 説明されているようにマシンセット カスタムリソース (CR) サンプルを含む新規 YAML ファイルを作成し、そのファイルに **<file\_name>.yaml** という名前を付けます。  
**<clusterID>** および **<role>** パラメーターの値を設定していることを確認します。
  - a. 特定のフィールドに設定する値が不明な場合は、クラスターから既存のマシンセットを確認できます。

```
$ oc get machinesets -n openshift-machine-api
```

### 出力例

NAME	DESIRED	CURRENT	READY	AVAILABLE	AGE
agl030519-vplxk-worker-us-east-1a	1	1	1	1	55m
agl030519-vplxk-worker-us-east-1b	1	1	1	1	55m

```

agl030519-vplxk-worker-us-east-1c 1      1      1      1      55m
agl030519-vplxk-worker-us-east-1d 0      0                      55m
agl030519-vplxk-worker-us-east-1e 0      0                      55m
agl030519-vplxk-worker-us-east-1f 0      0                      55m

```

- b. 特定のマシンセットの値を確認します。

```

$ oc get machineset <machineset_name> -n \
  openshift-machine-api -o yaml

```

### 出力例

```

...
template:
  metadata:
    labels:
      machine.openshift.io/cluster-api-cluster: agl030519-vplxk ❶
      machine.openshift.io/cluster-api-machine-role: worker ❷
      machine.openshift.io/cluster-api-machine-type: worker
      machine.openshift.io/cluster-api-machineset: agl030519-vplxk-worker-us-east-1a

```

- ❶ クラスター ID。  
❷ デフォルトのノードラベル。

2. 新規 **MachineSet** CR を作成します。

```

$ oc create -f <file_name>.yaml

```

3. マシンセットの一覧を表示します。

```

$ oc get machineset -n openshift-machine-api

```

### 出力例

```

NAME                                DESIRED  CURRENT  READY  AVAILABLE  AGE
agl030519-vplxk-infra-us-east-1a  1        1        1        1        11m
agl030519-vplxk-worker-us-east-1a  1        1        1        1        55m
agl030519-vplxk-worker-us-east-1b  1        1        1        1        55m
agl030519-vplxk-worker-us-east-1c  1        1        1        1        55m
agl030519-vplxk-worker-us-east-1d  0        0                      55m
agl030519-vplxk-worker-us-east-1e  0        0                      55m
agl030519-vplxk-worker-us-east-1f  0        0                      55m

```

新規のマシンセットが利用可能な場合、**DESIRED** および **CURRENT** の値は一致します。マシンセットが利用可能でない場合、数分待機してからコマンドを再度実行します。

## 1.2.2. 専用インフラストラクチャーノードの作成



## 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環境またはマスターノードがマシン API によって管理されているクラスターについて、Creating infrastructure machine set を参照してください。

クラスターの要件により、インフラストラクチャー (**infra** ノードとも呼ばれる) がプロビジョニングされます。インストーラーは、マスターノードとワーカーノードのプロビジョニングのみを提供します。ワーカーノードは、ラベル付けによって、インフラストラクチャーノードまたはアプリケーション (**app** と呼ばれる) として指定できます。

## 手順

1. アプリケーションノードとして機能させるワーカーノードにラベルを追加します。

```
$ oc label node <node-name> node-role.kubernetes.io/app=""
```

2. インフラストラクチャーノードとして機能する必要があるワーカーノードにラベルを追加します。

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

3. 該当するノードに **infra** ロールおよび **app** ロールがあるかどうかを確認します。

```
$ oc get nodes
```

4. ノードセクターのない Pod がデプロイされるノードのサブセット (ワーカーノードのデフォルトのデプロイメントなど) に割り当てられるようにデフォルトのノードセクターを作成します。たとえば、デフォルトでワーカーノードに Pod をデプロイするための **defaultNodeSelector** は以下ようになります。

```
defaultNodeSelector: node-role.kubernetes.io/app=
```

5. 新たにラベルが付けられた **infra** ノードにインフラストラクチャーリソースを移動します。

### 1.2.3. インフラストラクチャーマシンのマシン設定プール作成

インフラストラクチャーマシンに専用の設定が必要な場合は、infra プールを作成する必要があります。

## 手順

1. 特定のラベルを持つ infra ノードとして割り当てるノードに、ラベルを追加します。

```
$ oc label node <node_name> <label>
```

```
$ oc label node ci-ln-n8mqwr2-f76d1-xscn2-worker-c-6fmtx node-role.kubernetes.io/infra=
```

2. ワーカーロールとカスタムロールの両方をマシン設定セクターとして含まれるマシン設定プールを作成します。

```
$ cat infra.mcp.yaml
```



## 出力例

```

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

```

- ❶ ワーカーロールおよびカスタムロールを追加します。
- ❷ ノードに追加したラベルを **nodeSelector** として追加します。



## 注記

カスタムマシン設定プールは、ワーカープールからマシン設定を継承します。カスタムプールは、ワーカープールのターゲット設定を使用しますが、カスタムプールのみをターゲットに設定する変更をデプロイする機能を追加します。カスタムプールはワーカープールから設定を継承するため、ワーカープールへの変更もカスタムプールに適用されます。

3. YAML ファイルを用意した後に、マシン設定プールを作成できます。

```
$ oc create -f infra.mcp.yaml
```

4. マシン設定をチェックして、インフラストラクチャー設定が正常にレンダリングされていることを確認します。

```
$ oc get machineconfig
```

## 出力例

NAME	GENERATEDBYCONTROLLER
IGNITIONVERSION	CREATED
00-master	365c1cfd14de5b0e3b85e0fc815b0060f36ab955
2.2.0	31d
00-worker	365c1cfd14de5b0e3b85e0fc815b0060f36ab955
2.2.0	31d
01-master-container-runtime	
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	2.2.0 31d
01-master-kubelet	365c1cfd14de5b0e3b85e0fc815b0060f36ab955
2.2.0	31d
01-worker-container-runtime	
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	2.2.0 31d
01-worker-kubelet	365c1cfd14de5b0e3b85e0fc815b0060f36ab955
2.2.0	31d
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries	

365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	31d		
99-master-ssh		2.2.0	31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	31d		
99-worker-ssh		2.2.0	31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	19s		
rendered-master-072d4b2da7f88162636902b074e9e28e			
5b6fb8349a29735e48446d435962dec4547d3090 2.2.0	31d		
rendered-master-3e88ec72aed3886dec061df60d16d1af			
02c07496ba0417b3e12b78fb32baf6293d314f79 2.2.0	31d		
rendered-master-419bee7de96134963a15fdf9dd473b25			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	17d		
rendered-master-53f5c91c7661708adce18739cc0f40fb			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	13d		
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	7d3h		
rendered-master-dc7f874ec77fc4b969674204332da037			
5b6fb8349a29735e48446d435962dec4547d3090 2.2.0	31d		
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d			
5b6fb8349a29735e48446d435962dec4547d3090 2.2.0	31d		
rendered-worker-2640531be11ba43c61d72e82dc634ce6			
5b6fb8349a29735e48446d435962dec4547d3090 2.2.0	31d		
rendered-worker-4e48906dca84ee702959c71a53ee80e7			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	7d3h		
rendered-worker-4f110718fe88e5f349987854a1147755			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	17d		
rendered-worker-afc758e194d6188677eb837842d3b379			
02c07496ba0417b3e12b78fb32baf6293d314f79 2.2.0	31d		
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955 2.2.0	13d		

新規のマシン設定には、接頭辞 **rendered-infra-\*** が表示されるはずです。

- オプション: カスタムプールへの変更をデプロイするには、**infra** などのラベルとしてカスタムプール名を使用するマシン設定を作成します。これは必須ではありませんが、説明の目的でのみ表示されていることに注意してください。これにより、インフラストラクチャーノードのみに固有のカスタム設定を適用できます。



## 注記

新規マシン設定プールの作成後に、MCO はそのプールに新たにレンダリングされた設定を生成し、そのプールに関連付けられたノードは再起動して、新規設定を適用します。

- マシン設定を作成します。

```
$ cat infra.mc.yaml
```

## 出力例

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

```
labels:
  machineconfiguration.openshift.io/role: infra ❶
name: 51-infra
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:,infra
        filesystem: root
        mode: 0644
        path: /etc/infratest
```

❶ ノードに追加したラベルを **nodeSelector** として追加します。

b. マシン設定を infra のラベルが付いたノードに適用します。

```
$ oc create -f infra.mc.yaml
```

6. 新規のマシン設定プールが利用可能であることを確認します。

```
$ oc get mcp
```

### 出力例

NAME	CONFIG	UPDATED	UPDATING	DEGRADED	
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT			
DEGRADEDMACHINECOUNT	AGE				
infra	rendered-infra-60e35c2e99f42d976e084fa94da4d0fc	True	False	False	1
1	1	0	4m20s		
master	rendered-master-9360fdb895d4c131c7c4bebbae099c90	True	False	False	
3	3	3	0	91m	
worker	rendered-worker-60e35c2e99f42d976e084fa94da4d0fc	True	False	False	
2	2	2	0	91m	

この例では、ワーカーノードが infra ノードに変更されました。

### 関連情報

- カスタムプールでインフラマシンをグループ化する方法に関する詳細は、[Node configuration management with machine config pools](#) を参照してください。

## 1.3. マシンセットリソースのインフラストラクチャーノードへの割り当て

インフラストラクチャーマシンセットの作成後、**worker** および **infra** ロールが新規の infra ノードに適用されます。**infra** ロールが割り当てられたノードは、**worker** ロールも適用されている場合でも、環境を実行するために必要なサブスクリプションの合計数にはカウントされません。

ただし、infra ノードに worker ロールが割り当てられている場合は、ユーザーのワークロードが誤って infra ノードに割り当てられる可能性があります。これを回避するには、テイントを、制御する必要のある Pod の infra ノードおよび容認に適用できます。

### 1.3.1. テイントおよび容認を使用したインフラストラクチャーノードのワークロードのバインディング

**infra** および **worker** ロールが割り当てられている infra ノードがある場合、ユーザーのワークロードがこれに割り当てられないようにノードを設定する必要があります。



#### 重要

infra ノード用に作成されたデュアル **infra,worker** ラベルを保持し、テイントおよび容認 (Toleration) を使用してユーザーのワークロードがスケジュールされているノードを管理することを推奨します。ノードから **worker** ラベルを削除する場合には、カスタムプールを作成して管理する必要があります。**master** または **worker** 以外のラベルが割り当てられたノードは、カスタムプールなしには MCO で認識されません。**worker** ラベルを維持すると、カスタムラベルを選択するカスタムプールが存在しない場合に、ノードをデフォルトのワーカーマシン設定プールで管理できます。**infra** ラベルは、サブスクリプションの合計数にカウントされないクラスターと通信します。

#### 前提条件

- 追加の **MachineSet** を OpenShift Container Platform クラスターに設定します。

#### 手順

1. テイントを infra ノードに追加し、ユーザーのワークロードをこれにスケジュールできないようにします。
  - a. ノードにテイントがあるかどうかを判別します。

```
$ oc describe nodes <node_name>
```

#### 出力例

```
oc describe node ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
Name:          ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
Roles:         worker
...
Taints:        node-role.kubernetes.io/infra:NoSchedule
...
```

この例では、ノードにテイントがあることを示しています。次の手順に進み、容認を Pod に追加してください。

- b. ユーザーワークロードをスケジュールリングできないように、テイントを設定していない場合は、以下を実行します。

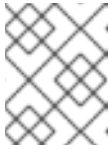
```
$ oc adm taint nodes <node_name> <key>:<effect>
```

以下に例を示します。

```
$ oc adm taint nodes node1 node-role.kubernetes.io/infra:NoSchedule
```

この例では、テイントを、キー **node-role.kubernetes.io/infra** およびテイントの effect **NoSchedule** を持つ **node1** に配置します。effect が **NoSchedule** のノードは、テイントを容認する Pod のみをスケジュールしますが、既存の Pod はノードにスケジュールされた

ままになります。



### 注記

Descheduler が使用されると、ノードのテイントに違反する Pod はクラスターからエビクトされる可能性があります。

2. ルーター、レジストリーおよびモニタリングのワークロードなどの、infra ノードにスケジュールする必要のある Pod 設定の容認を追加します。以下のコードを **Pod** オブジェクトの仕様に追加します。

```
tolerations:
  - effect: NoSchedule ❶
    key: node-role.kubernetes.io/infra ❷
    operator: Exists ❸
```

- ❶ ノードに追加した effect を指定します。
- ❷ ノードに追加したキーを指定します。
- ❸ **Exists** Operator を、キー **node-role.kubernetes.io/infra** のあるテイントがノードに存在するように指定します。

この容認は、**oc adm taint** コマンドで作成されたテイントと一致します。この容認のある Pod は infra ノードにスケジュールできます。



### 注記

OLM でインストールされた Operator の Pod を infra ノードに常に移動できる訳ではありません。Operator Pod を移動する機能は、各 Operator の設定によって異なります。

3. スケジューラーを使用して Pod を infra ノードにスケジュールします。詳細は、**Pod のノードへの配置の制御** についてのドキュメントを参照してください。

## 関連情報

- ノードへの Pod のスケジューリングに関する一般的な情報については、[Controlling pod placement using the scheduler](#) を参照してください。

## 1.4. リソースのインフラストラクチャマシンセットへの移行

インフラストラクチャーリソースの一部はデフォルトでクラスターにデプロイされます。それらは、作成したインフラストラクチャマシンセットに移行できます。

### 1.4.1. 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
router-default-955d875f4-255g8	0/1	Terminating	0	19h	10.129.2.4	ip-10-0-148-172.ec2.internal

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> ❶
```

- ❶ 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。

## 1.4.2. デフォルトレジストリーの移行

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

## 前提条件

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

## 手順

1. **config/instance** オブジェクトを表示します。

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

## 出力例

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12fdee2931b
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。

### 1.4.3. モニターリングソリューションの移動

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

#### 手順

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

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:

```



```

config.yaml: |+
  alertmanagerMain:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  prometheusK8s:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  prometheusOperator:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  grafana:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  k8sPrometheusAdapter:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  kubeStateMetrics:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  telemeterClient:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  openshiftStateMetrics:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
  thanosQuerier:
    nodeSelector:
      node-role.kubernetes.io/infra: ""

```

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

2. 新規の設定マップを適用します。

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

3. モニタリング Pod が新規マシンに移行することを確認します。

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

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

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

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

#### 1.4.4. クラスターロギングリソースの移動

すべてのクラスターロギングコンポーネント、Elasticsearch、Kibana、および Curator の Pod を異なるノードにデプロイするように Cluster Logging Operator を設定できます。Cluster Logging Operator Pod については、インストールされた場所から移動することはできません。

たとえば、Elasticsearch Pod の CPU、メモリーおよびディスクの要件が高いために、この Pod を別のノードに移動できます。

## 前提条件

- クラスターロギングおよび Elasticsearch がインストールされている。これらの機能はデフォルトでインストールされません。

## 手順

1. **openshift-logging** プロジェクトで **ClusterLogging** カスタムリソース (CR) を編集します。

```
$ oc edit ClusterLogging instance
```

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:
  collection:
    logs:
      fluentd:
        resources: null
        type: fluentd
  curation:
    curator:
      nodeSelector: ❶
      node-role.kubernetes.io/infra: "
      resources: null
      schedule: 30 3 * * *
      type: curator
  logStore:
    elasticsearch:
      nodeCount: 3
      nodeSelector: ❷
      node-role.kubernetes.io/infra: "
      redundancyPolicy: SingleRedundancy
      resources:
        limits:
          cpu: 500m
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
      storage: {}
      type: elasticsearch
  managementState: Managed
  visualization:
    kibana:
      nodeSelector: ❸
      node-role.kubernetes.io/infra: "
      proxy:
        resources: null
      replicas: 1
      resources: null
```

```
type: kibana
```

```
...
```

- 1 2 3** 適切な値が設定された **nodeSelector** パラメーターを、移動する必要があるコンポーネントに追加します。表示されている形式の **nodeSelector** を使用することも、ノードに指定された値に基づいて **<key>: <value>** ペアを使用することもできます。

## 検証

コンポーネントが移動したことを確認するには、**oc get pod -o wide** コマンドを使用できます。

以下に例を示します。

- Kibana Pod を **ip-10-0-147-79.us-east-2.compute.internal** ノードから移動する必要がある場合、以下を実行します。

```
$ oc get pod kibana-5b8bdf44f9-ccpq9 -o wide
```

### 出力例

```
NAME                                READY STATUS RESTARTS AGE IP          NODE
NOMINATED NODE READINESS GATES
kibana-5b8bdf44f9-ccpq9 2/2    Running 0      27s 10.129.2.18 ip-10-0-147-79.us-east-2.compute.internal <none> <none>
```

- Kibana Pod を、専用インフラストラクチャーノードである **ip-10-0-139-48.us-east-2.compute.internal** ノードに移動する必要がある場合、以下を実行します。

```
$ oc get nodes
```

### 出力例

```
NAME                                STATUS ROLES    AGE VERSION
ip-10-0-133-216.us-east-2.compute.internal Ready master    60m v1.18.3
ip-10-0-139-146.us-east-2.compute.internal Ready master    60m v1.18.3
ip-10-0-139-192.us-east-2.compute.internal Ready worker    51m v1.18.3
ip-10-0-139-241.us-east-2.compute.internal Ready worker    51m v1.18.3
ip-10-0-147-79.us-east-2.compute.internal Ready worker    51m v1.18.3
ip-10-0-152-241.us-east-2.compute.internal Ready master    60m v1.18.3
ip-10-0-139-48.us-east-2.compute.internal Ready infra     51m v1.18.3
```

ノードには **node-role.kubernetes.io/infra: "** ラベルがあることに注意してください。

```
$ oc get node ip-10-0-139-48.us-east-2.compute.internal -o yaml
```

### 出力例

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-139-48.us-east-2.compute.internal
```

```

selfLink: /api/v1/nodes/ip-10-0-139-48.us-east-2.compute.internal
uid: 62038aa9-661f-41d7-ba93-b5f1b6ef8751
resourceVersion: '39083'
creationTimestamp: '2020-04-13T19:07:55Z'
labels:
  node-role.kubernetes.io/infra: "
...

```

- Kibana Pod を移動するには、**ClusterLogging** CR を編集してノードセクターを追加します。

```

apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:

...

visualization:
  kibana:
    nodeSelector: ❶
    node-role.kubernetes.io/infra: "
    proxy:
      resources: null
    replicas: 1
    resources: null
    type: kibana

```

- ❶ ノード仕様のラベルに一致するノードセクターを追加します。

- CR を保存した後に、現在の Kibana Pod は終了し、新規 Pod がデプロイされます。

```
$ oc get pods
```

## 出力例

NAME	READY	STATUS	RESTARTS	AGE
cluster-logging-operator-84d98649c4-zb9g7	1/1	Running	0	29m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg	2/2	Running	0	28m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj	2/2	Running	0	28m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78	2/2	Running	0	28m
fluentd-42dzz	1/1	Running	0	28m
fluentd-d74rq	1/1	Running	0	28m
fluentd-m5vr9	1/1	Running	0	28m
fluentd-nkxl7	1/1	Running	0	28m
fluentd-pdvqb	1/1	Running	0	28m
fluentd-tflh6	1/1	Running	0	28m
kibana-5b8bdf44f9-ccpq9	2/2	Terminating	0	4m11s
kibana-7d85dcffc8-bfpfp	2/2	Running	0	33s

- 新規 Pod が **ip-10-0-139-48.us-east-2.compute.internal** ノードに置かれます。

```
$ oc get pod kibana-7d85dcffc8-bfpfp -o wide
```

### 出力例

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE READINESS GATES						
kibana-7d85dcffc8-bfpfp	2/2	Running	0	43s	10.131.0.22	ip-10-0-139-48.us-east-2.compute.internal
	<none>	<none>	<none>			

- しばらくすると、元の Kibana Pod が削除されます。

```
$ oc get pods
```

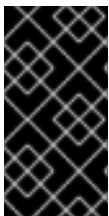
### 出力例

NAME	READY	STATUS	RESTARTS	AGE
cluster-logging-operator-84d98649c4-zb9g7	1/1	Running	0	30m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg	2/2	Running	0	29m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj	2/2	Running	0	29m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78	2/2	Running	0	29m
fluentd-42dzz	1/1	Running	0	29m
fluentd-d74rq	1/1	Running	0	29m
fluentd-m5vr9	1/1	Running	0	29m
fluentd-nkx17	1/1	Running	0	29m
fluentd-pdvqb	1/1	Running	0	29m
fluentd-tflh6	1/1	Running	0	29m
kibana-7d85dcffc8-bfpfp	2/2	Running	0	62s

## 1.5. CLUSTER AUTOSCALER について

Cluster Autoscaler は、現行のデプロイメントのニーズに合わせて OpenShift Container Platform クラスターのサイズを調整します。これは、Kubernetes 形式の宣言引数を使用して、特定のクラウドプロバイダーのオブジェクトに依存しないインフラストラクチャー管理を提供します。Cluster Autoscaler には cluster スcopeがあり、特定の namespace には関連付けられていません。

Cluster Autoscaler は、リソース不足のために現在のノードのいずれにもスケジュールできない Pod がある場合や、デプロイメントのニーズを満たすために別のノードが必要な場合に、クラスターのサイズを拡大します。Cluster Autoscaler は、指定される制限を超えてクラスターリソースを拡大することはありません。



### 重要

作成する **ClusterAutoscaler** リソース定義の **maxNodesTotal** 値が、クラスター内のマシンの想定される合計数に対応するのに十分な大きさの値であることを確認します。この値は、コントロールプレーンマシンの数とスケーリングする可能性のあるコンピュータマシンの数に対応できる値である必要があります。

Cluster Autoscaler は、リソースの使用量が少なく、重要な Pod すべてが他のノードに適合する場合など、一部のノードが長い期間にわたって不要な状態が続く場合にクラスターのサイズを縮小します。

以下のタイプの Pod がノードにある場合、Cluster Autoscaler はそのノードを削除しません。

- 制限のある Pod の Disruption Budget (停止状態の予算、PDB) を持つ Pod。

- デフォルトでノードで実行されない Kube システム Pod。
- PDB を持たないか、または制限が厳しい PDB を持つ Kuber システム Pod。
- デプロイメント、レプリカセット、またはステートフルセットなどのコントローラーオブジェクトによってサポートされない Pod。
- ローカルストレージを持つ Pod。
- リソース不足、互換性のないノードセクターまたはアフィニティー、一致する非アフィニティーなどにより他の場所に移動できない Pod。
- それらに **"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"** アノテーションがない場合、**"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"** アノテーションを持つ Pod。

Cluster Autoscaler を設定する場合、使用に関する追加の制限が適用されます。

- 自動スケーリングされたノードグループにあるノードを直接変更しない。同じノードグループ内のすべてのノードには同じ容量およびラベルがあり、同じシステム Pod を実行します。
- Pod の要求を指定します。
- Pod がすぐに削除されるのを防ぐ必要がある場合、適切な PDB を設定します。
- クラウドプロバイダーのクォータが、設定する最大のノードプールに対応できる十分な大きさであることを確認します。
- クラウドプロバイダーで提供されるものなどの、追加のノードグループの Autoscaler を実行しない。

Horizontal Pod Autoscaler (HPA) および Cluster Autoscaler は複数の異なる方法でクラスターリソースを変更します。HPA は、現在の CPU 負荷に基づいてデプロイメント、またはレプリカセットのレプリカ数を変更します。負荷が増大すると、HPA はクラスターで利用できるリソース量に関係なく、新規レプリカを作成します。十分なリソースがない場合、Cluster Autoscaler はリソースを追加し、HPA で作成された Pod が実行できるようにします。負荷が減少する場合、HPA は一部のレプリカを停止します。この動作によって一部のノードの使用率が低くなるか、または完全に空になる場合、Cluster Autoscaler は不必要なノードを削除します。

Cluster Autoscaler は Pod の優先順位を考慮に入れます。Pod の優先順位とプリエンプション機能により、クラスターに十分なリソースがない場合に優先順位に基づいて Pod のスケジューリングを有効にできますが、Cluster Autoscaler はクラスターがすべての Pod を実行するのに必要なリソースを確保できます。これら両方の機能の意図を反映するべく、Cluster Autoscaler には優先順位のカットオフ機能が含まれています。このカットオフを使用して Best Effort の Pod をスケジュールできますが、これにより Cluster Autoscaler がリソースを増やすことはなく、余分なリソースがある場合にのみ実行されます。

カットオフ値よりも低い優先順位を持つ Pod は、クラスターをスケールアップせず、クラスターのスケールダウンを防ぐこともありません。これらの Pod を実行するために新規ノードは追加されず、これらの Pod を実行しているノードはリソースを解放するために削除される可能性があります。

### 1.5.1. ClusterAutoscaler リソース定義

この **ClusterAutoscaler** リソース定義は、Cluster Autoscaler のパラメーターおよびサンプル値を表示します。

```
apiVersion: "autoscaling.openshift.io/v1"
```

```

kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10 ❶
  resourceLimits:
    maxNodesTotal: 24 ❷
    cores:
      min: 8 ❸
      max: 128 ❹
    memory:
      min: 4 ❺
      max: 256 ❻
    gpus:
      - type: nvidia.com/gpu ❷
        min: 0 ❸
        max: 16 ❹
      - type: amd.com/gpu ❷
        min: 0 ❸
        max: 4 ❹
  scaleDown: ❸
    enabled: true ❸
    delayAfterAdd: 10m ❸
    delayAfterDelete: 5m ❸
    delayAfterFailure: 30s ❸
    unneededTime: 5m ❸

```

- ❶ Cluster Autoscaler に追加のノードをデプロイさせるために Pod が超えている必要のある優先順位を指定します。32 ビットの整数値を入力します。**podPriorityThreshold** 値は、各 Pod に割り当てた **PriorityClass** の値と比較されます。
- ❷ デプロイするノードの最大数を指定します。この値は、Autoscaler が制御するマシンだけでなく、クラスターにデプロイされるマシンの合計数です。この値は、すべてのコントロールプレーンおよびコンピュータマシン、および **MachineAutoscaler** リソースに指定するレプリカの合計数に対応するのに十分な大きさの値であることを確認します。
- ❸ クラスターにデプロイするコアの最小数を指定します。
- ❹ クラスターにデプロイするコアの最大数を指定します。
- ❺ クラスターのメモリーの最小量 (GiB 単位) を指定します。
- ❻ クラスターのメモリーの最大量 (GiB 単位) を指定します。
- ❷ ❸ オプションで、デプロイする GPU ノードのタイプを指定します。**nvidia.com/gpu** および **amd.com/gpu** のみが有効なタイプです。
- ❸ ❸ クラスターにデプロイする GPU の最小数を指定します。
- ❸ ❸ クラスターにデプロイする GPU の最大数を指定します。
- ❸ このセクションでは、有効な **ParseDuration** 期間 (**ns**、**us**、**ms**、**s**、**m**、および **h** を含む) を使用して各アクションについて待機する期間を指定できます。

- 14 Cluster Autoscaler が不必要なノードを削除できるかどうかを指定します。
- 15 オプションで、ノードが最後に **追加** されてからノードを削除するまで待機する期間を指定します。値を指定しない場合、デフォルト値の **10m** が使用されます。
- 16 ノードが最後に **削除** されたからノードを削除するまで待機する期間を指定します。値を指定しない場合、デフォルト値の **10s** が使用されます。
- 17 スケールダウンが失敗してからノードを削除するまで待機する期間を指定します。値を指定しない場合、デフォルト値の **3m** が使用されます。
- 18 不要なノードが削除の対象となるまでの期間を指定します。値を指定しない場合、デフォルト値の **10m** が使用されます。



### 注記

スケーリング操作の実行時に、Cluster Autoscaler は、デプロイするコアの最小および最大数、またはクラスター内のメモリー量などの **ClusterAutoscaler** リソース定義に設定された範囲内に残ります。ただし、Cluster Autoscaler はそれらの範囲内に留まるようクラスターの現在の値を修正しません。

## 1.5.2. Cluster Autoscaler のデプロイ

Cluster Autoscaler をデプロイするには、**ClusterAutoscaler** リソースのインスタンスを作成します。

### 手順

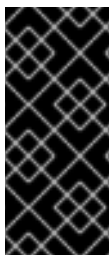
1. カスタマイズされたリソース定義を含む **ClusterAutoscaler** リソースの YAML ファイルを作成します。
2. クラスターにリソースを作成します。

```
$ oc create -f <filename>.yaml 1
```

- 1 **<filename>** は、カスタマイズしたリソースファイルの名前です。

## 1.6. MACHINE AUTOSCALER

Machine Autoscaler は、マシンセットで OpenShift Container Platform クラスターにデプロイするマシン数を調整します。デフォルトの **worker** マシンセットおよび作成する他のマシンセットの両方をスケーリングできます。Machine Autoscaler は、追加のデプロイメントをサポートするのに十分なリソースがクラスターにない場合に追加のマシンを作成します。**MachineAutoscaler** リソースの値への変更 (例: インスタンスの最小または最大数) は、それらがターゲットとするマシンセットに即時に適用されます。



### 重要

マシンをスケーリングするには、Cluster Autoscaler の Machine Autoscaler をデプロイする必要があります。Cluster Autoscaler は、スケーリングできるリソースを判別するために、Machine Autoscaler が設定するアノテーションをマシンセットで使用します。Machine Autoscaler を定義せずにクラスター Autoscaler を定義する場合、クラスター Autoscaler はクラスターをスケーリングできません。



### 1.6.1. MachineAutoscaler リソース定義

この **MachineAutoscaler** リソース定義は、Machine Autoscaler のパラメーターおよびサンプル値を表示します。

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
  name: "worker-us-east-1a" ❶
  namespace: "openshift-machine-api"
spec:
  minReplicas: 1 ❷
  maxReplicas: 12 ❸
  scaleTargetRef: ❹
    apiVersion: machine.openshift.io/v1beta1
    kind: MachineSet ❺
    name: worker-us-east-1a ❻
```

- ❶ Machine Autoscaler の名前を指定します。この Machine Autoscaler がスケーリングするマシンセットを簡単に特定できるようにするには、スケーリングするマシンセットの名前を指定するか、またはこれを組み込みます。マシンセットの名前は以下の形式を取ります。 **<clusterid>-<machineset>-<aws-region-az>**
- ❷ クラスター Autoscaler がクラスターのスケーリングを開始した後に、指定されたゾーンに残っている必要のある指定されたタイプのマシンの最小数を指定します。AWS、GCP、または Azure で実行している場合は、この値は **0** に設定できます。他のプロバイダーの場合は、この値は **0** に設定しないでください。
- ❸ Cluster Autoscaler がクラスタースケリングの開始後に指定された AWS ゾーンにデプロイできる指定されたタイプのマシンの最大数を指定します。**ClusterAutoscaler** リソース定義の **maxNodesTotal** 値が、Machine AutoScaler がこの数のマシンをデプロイするのに十分な大きさの値であることを確認します。
- ❹ このセクションでは、スケーリングする既存のマシンセットを記述する値を指定します。
- ❺ **kind** パラメーターの値は常に **MachineSet** です。
- ❻ **name** の値は、**metadata.name** パラメーター値に示されるように既存のマシンセットの名前に一致する必要があります。

### 1.6.2. Machine Autoscaler のデプロイ

Machine Autoscaler をデプロイするには、**MachineAutoscaler** リソースのインスタンスを作成します。

#### 手順

1. カスタマイズされたリソース定義を含む **MachineAutoscaler** リソースの YAML ファイルを作成します。
2. クラスターにリソースを作成します。

```
$ oc create -f <filename>.yaml ❶
```

- 1 <filename> は、カスタマイズしたリソースファイルの名前です。

## 1.7. FEATUREGATE の使用によるテクノロジープレビュー機能の有効化

**FeatureGate** カスタムリソース (CR) を編集して、クラスターのすべてのノードに対して現在のテクノロジープレビュー機能のサブセットをオンにすることができます。

## 1.8. ETCD タスク

etcd のバックアップ、etcd 暗号化の有効化または無効化、または etcd データのデフラグを行います。

### 1.8.1. etcd 暗号化について

デフォルトで、etcd データは OpenShift Container Platform で暗号化されません。クラスターの etcd 暗号化を有効にして、データセキュリティの層を追加で提供することができます。たとえば、etcd バックアップが正しくない公開先に公開される場合に機密データが失われるように保護することができます。

etcd の暗号化を有効にすると、以下の OpenShift API サーバーおよび Kubernetes API サーバーリソースが暗号化されます。

- シークレット
- 設定マップ
- ルート
- OAuth アクセストークン
- OAuth 認証トークン

etcd 暗号を有効にすると、暗号化キーが作成されます。これらのキーは週ごとにローテーションされます。etcd バックアップから復元するには、これらのキーが必要です。

### 1.8.2. etcd 暗号化の有効化

etcd 暗号化を有効にして、クラスターで機密性の高いリソースを暗号化できます。



#### 警告

初期の暗号化プロセスが完了するまでは、etcd のバックアップを取ることは推奨されません。暗号化プロセスが完了しない場合、バックアップは部分的にのみ暗号化される可能性があります。

#### 前提条件

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

## 手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **encryption** フィールドタイプを **aescbc** に設定します。

```
spec:
  encryption:
    type: aescbc ❶
```

- ❶ **aescbc** タイプは、暗号化を実行するために PKCS#7 パディングを実装している AES-CBC と 32 バイトのキーが使用されることを意味します。

3. 変更を適用するためにファイルを保存します。  
暗号化プロセスが開始されます。クラスターのサイズによっては、このプロセスが完了するまで 20 分以上かかる場合があります。

4. etcd 暗号化が正常に行われたことを確認します。

- a. OpenShift API サーバーの **Encrypted** ステータスを確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io, oauthaccesstokens.oauth.openshift.io,
oauthauthorizetokens.oauth.openshift.io
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

- b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

### 1.8.3. etcd 暗号化の無効化

クラスターで etcd データの暗号化を無効にできます。

## 前提条件

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

## 手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **encryption** フィールドタイプを **identity** に設定します。

```
spec:
  encryption:
    type: identity 1
```

- 1 **identity** タイプはデフォルト値であり、暗号化は実行されないことを意味します。

3. 変更を適用するためにファイルを保存します。  
復号化プロセスが開始されます。クラスターのサイズによっては、このプロセスが完了するまで 20 分以上かかる場合があります。
4. etcd の復号化が正常に行われたことを確認します。
  - a. OpenShift API サーバーの **Encrypted** ステータス条件を確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

- b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に復号化されたことを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

### 1.8.4. etcd データのバックアップ

以下の手順に従って、etcd スナップショットを作成し、静的 Pod のリソースをバックアップして etcd データをバックアップします。このバックアップは保存でき、etcd を復元する必要がある場合に後で使用することができます。



### 重要

単一マスターホストからのバックアップのみを保存します。クラスター内の各マスターホストからのバックアップは取りません。

### 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。
- クラスター全体のプロキシが有効になっているかどうかを確認している。

### ヒント

**oc get proxy cluster -o yaml** の出力を確認して、プロキシが有効にされているかどうかを確認できます。プロキシは、**httpProxy**、**httpsProxy**、および **noProxy** フィールドに値が設定されている場合に有効にされます。

### 手順

1. マスターノードのデバッグセッションを開始します。

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

2. ルートディレクトリーをホストに切り替えます。

```
sh-4.2# chroot /host
```

3. クラスター全体のプロキシが有効になっている場合は、**NO\_PROXY**、**HTTP\_PROXY**、および **HTTPS\_PROXY** 環境変数をエクスポートしていることを確認します。
4. **etcd-snapshot-backup.sh** スクリプトを実行し、バックアップの保存先となる場所を渡します。

### ヒント

**cluster-backup.sh** スクリプトは etcd Cluster Operator のコンポーネントとして維持され、**etcdctl snapshot save** コマンドに関連するラッパーです。

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

### スクリプトの出力例

```
1bf371f1b5a483927cd01bb593b0e12cff406eb8d7d0acf4ab079c36a0abd3f7
etcdctl version: 3.3.18
API version: 3.3
found latest kube-apiserver-pod: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-7
found latest kube-controller-manager-pod: /etc/kubernetes/static-pod-resources/kube-
controller-manager-pod-8
```

```
found latest kube-scheduler-pod: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd-pod: /etc/kubernetes/static-pod-resources/etcd-pod-2
Snapshot saved at /home/core/assets/backup/snapshot_2020-03-18_220218.db
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

この例では、マスターホストの **/home/core/assets/backup/** ディレクトリーにファイルが2つ作成されます。

- **snapshot\_<datetimestamp>.db**: このファイルは etcd スナップショットです。
- **static\_kuberesources\_<datetimestamp>.tar.gz**: このファイルには、静的 Pod のリソースが含まれます。etcd 暗号化が有効にされている場合、etcd スナップショットの暗号化キーも含まれます。



### 注記

etcd 暗号化が有効にされている場合、セキュリティ上の理由から、この2つ目のファイルを etcd スナップショットとは別に保存することが推奨されます。ただし、このファイルは etcd スナップショットから復元するために必要になります。

etcd 暗号化はキーではなく値のみを暗号化することに注意してください。つまり、リソースタイプ、namespace、およびオブジェクト名は暗号化されません。

## 1.8.5. 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行う必要があります。

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 | 251cd444483d811c3 | 3.4.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.4.9 | 41 MB | false | false |
7 | 91624 | 91624 | | 1
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.4.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+

```

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

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

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

- a. **NOSPACE** アラームがあるかどうかを確認します。

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



## 出力例

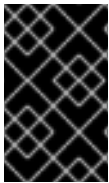
```
memberID:12345678912345678912 alarm:NOSPACE
```

- b. アラームをクリアします。

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

## 1.8.6. クラスターの直前の状態への復元

保存された etcd バックアップを使用して、クラスターの以前の状態に戻すことができます。etcd バックアップを使用して単一のコントロールパネルホストを復元します。次に、etcd クラスター Operator は残りのマスターホストへのスケーリングを処理します。



## 重要

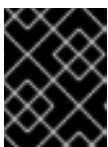
クラスターを復元する際に、同じ z-stream リリースから取得した etcd バックアップを使用する必要があります。たとえば、OpenShift Container Platform 4.5.2 クラスターは、4.5.2 から取得した etcd バックアップを使用する必要があります。

## 前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- リカバリーホストとして使用する正常なマスターホスト。
- マスターホストへの SSH アクセス。
- etcd スナップショットと静的 Pod のリソースの両方を含むバックアップディレクトリー (同じバックアップから取られるもの)。ディレクトリー内のファイル名は、**snapshot\_<timestamp>.db** および **static\_kuberesources\_<timestamp>.tar.gz** の形式にする必要があります。

## 手順

1. リカバリーホストとして使用するコントロールプレーンホストを選択します。これは、復元操作を実行するホストです。
2. リカバリーホストを含む、各コントロールプレーンノードへの SSH 接続を確立します。Kubernetes API サーバーは復元プロセスの開始後にアクセスできなくなるため、コントロールプレーンノードにはアクセスできません。このため、別のターミナルで各コントロールプレーンホストに SSH 接続を確立することが推奨されます。

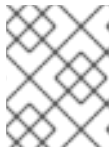


## 重要

この手順を完了しないと、復元手順を完了するためにマスターホストにアクセスすることができなくなり、この状態からクラスターを回復できなくなります。

3. etcd バックアップディレクトリーをリカバリーコントロールプレーンホストにコピーします。この手順では、etcd スナップショットおよび静的 Pod のリソースを含む **backup** ディレクトリーを、リカバリーコントロールプレーンホストの **/home/core/** ディレクトリーにコピーしていることを前提としています。

4. 他のすべてのコントロールプレーンノードで静的 Pod を停止します。



### 注記

リカバリーホストで Pod を手動で停止する必要はありません。リカバリースクリプトは、リカバリーホストの Pod を停止します。

- a. リカバリーホストではないコントロールプレーンホストにアクセスします。
- b. 既存の etcd Pod ファイルを kubelet マニフェストディレクトリーから移動します。

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

- c. etcd Pod が停止していることを確認します。

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

コマンドの出力は空であるはずです。空でない場合は、数分待機してから再度確認します。

- d. 既存の Kubernetes API サーバー Pod ファイルを kubelet マニフェストディレクトリーから移動します。

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

- e. Kubernetes API サーバー Pod が停止していることを確認します。

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep kube-apiserver | grep -v operator
```

コマンドの出力は空であるはずです。空でない場合は、数分待機してから再度確認します。

- f. etcd データディレクトリーを別の場所に移動します。

```
[core@ip-10-0-154-194 ~]$ sudo mv /var/lib/etcd/ /tmp
```

- g. リカバリーホストではない他のマスターホストでこの手順を繰り返します。

5. リカバリーコントロールプレーンホストにアクセスします。
6. クラスター全体のプロキシが有効になっている場合は、**NO\_PROXY**、**HTTP\_PROXY**、および **HTTPS\_PROXY** 環境変数をエクスポートしていることを確認します。

### ヒント

**oc get proxy cluster -o yaml** の出力を確認して、プロキシが有効にされているかどうかを確認できます。プロキシは、**httpProxy**、**httpsProxy**、および **noProxy** フィールドに値が設定されている場合に有効にされます。

7. リカバリーコントロールプレーンホストで復元スクリプトを実行し、パスを etcd バックアップディレクトリーに渡します。

```
[core@ip-10-0-143-125 ~]$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/backup
```

## スクリプトの出力例

```
...stopping kube-scheduler-pod.yaml
...stopping kube-controller-manager-pod.yaml
...stopping etcd-pod.yaml
...stopping kube-apiserver-pod.yaml
Waiting for container etcd to stop
.complete
Waiting for container etcdctl to stop
.....complete
Waiting for container etcd-metrics to stop
complete
Waiting for container kube-controller-manager to stop
complete
Waiting for container kube-apiserver to stop
.....complete
Waiting for container kube-scheduler to stop
complete
Moving etcd data-dir /var/lib/etcd/member to /var/lib/etcd-backup
starting restore-etcd static pod
starting kube-apiserver-pod.yaml
static-pod-resources/kube-apiserver-pod-7/kube-apiserver-pod.yaml
starting kube-controller-manager-pod.yaml
static-pod-resources/kube-controller-manager-pod-7/kube-controller-manager-pod.yaml
starting kube-scheduler-pod.yaml
static-pod-resources/kube-scheduler-pod-8/kube-scheduler-pod.yaml
```

8. すべてのマスターホストで kubelet サービスを再起動します。

- a. リカバリーホストから以下のコマンドを実行します。

```
[core@ip-10-0-143-125 ~]$ sudo systemctl restart kubelet.service
```

- b. 他のすべてのマスターホストでこの手順を繰り返します。

9. 単一メンバーのコントロールプレーンが正常に起動していることを確認します。

- a. リカバリーホストから etcd コンテナが実行中であることを確認します。

```
[core@ip-10-0-143-125 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

## 出力例

```
3ad41b7908e32
36f86e2eeaafe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running           etcd              0
7c05f8af362f0
```

- b. リカバリーホストから、etcd Pod が実行されていることを確認します。

```
[core@ip-10-0-143-125 ~]$ oc get pods -n openshift-etcd | grep etcd
```



## 注記

このコマンドを実行する前に **oc login** の実行を試行し、以下のエラーを受信すると、認証コントローラーが起動し、再試行するまでしばらく待機します。

```
Unable to connect to the server: EOF
```

## 出力例

```
NAME                                READY STATUS  RESTARTS AGE
etcd-ip-10-0-143-125.ec2.internal  1/1   Running    1      2m47s
```

ステータスが **Pending** の場合や出力に複数の実行中の etcd Pod が一覧表示される場合、数分待機してから再度チェックを行います。

- etcd の再デプロイメントを強制的に実行します。

クラスターにアクセスできるターミナルで、**cluster-admin** ユーザーとして以下のコマンドを実行します。

```
$ oc patch etcd cluster -p='{ "spec": { "forceRedeploymentReason": "recovery-"$( date --rfc-3339=ns )"' } }' --type=merge ①
```

- ① **forceRedeploymentReason** 値は一意である必要があります。そのため、タイムスタンプが付加されます。

etcd クラスター Operator が再デプロイメントを実行すると、初期ブートストラップのスケールアップと同様に、既存のノードが新規 Pod と共に起動します。

- すべてのノードが最新のリリースに更新されていることを確認します。

クラスターにアクセスできるターミナルで、**cluster-admin** ユーザーとして以下のコマンドを実行します。

```
$ oc get etcd -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

etcd の **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリリースにあることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 7 ①
```

- ① この例では、最新のリリース番号は 7 です。

出力に **2 nodes are at revision 6; 1 nodes are at revision 7** などの複数のリリース番号が含まれる場合、これは更新が依然として進行中であることを意味します。数分待機した後に再試行します。

- etcd の再デプロイ後に、コントロールプレーンの新規ロールアウトを強制的に実行します。kubelet が内部ロードバランサーを使用して API サーバーに接続されているため、Kubernetes API サーバーは他のノードに再インストールされます。

クラスターにアクセスできるターミナルで、**cluster-admin** ユーザーとして以下のコマンドを実行します。

a. **kubeapiserver** を更新します。

```
$ oc patch kubeapiserver cluster -p='{ "spec": { "forceRedeploymentReason": "recovery-"'$( date --rfc-3339=ns )'" }}" --type=merge
```

すべてのノードが最新のリビジョンに更新されていることを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{ "\n" }{.message}{ "\n" }
```

**NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

**1** この例では、最新のリビジョン番号は **7** です。

出力に **2 nodes are at revision 6; 1 nodes are at revision 7** などの複数のリビジョン番号が含まれる場合、これは更新が依然として進行中であることを意味します。数分待機した後、再試行します。

b. **kubecontrollermanager** を更新します。

```
$ oc patch kubecontrollermanager cluster -p='{ "spec": { "forceRedeploymentReason": "recovery-"'$( date --rfc-3339=ns )'" }}" --type=merge
```

すべてのノードが最新のリビジョンに更新されていることを確認します。

```
$ oc get kubecontrollermanager -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{ "\n" }{.message}{ "\n" }
```

**NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

**1** この例では、最新のリビジョン番号は **7** です。

出力に **2 nodes are at revision 6; 1 nodes are at revision 7** などの複数のリビジョン番号が含まれる場合、これは更新が依然として進行中であることを意味します。数分待機した後、再試行します。

c. **kubescheduler** を更新します。

```
$ oc patch kubescheduler cluster -p='{ "spec": { "forceRedeploymentReason": "recovery-$( date --rfc-3339=ns )"' --type=merge
```

すべてのノードが最新のリビジョンに更新されていることを確認します。

```
$ oc get kubescheduler -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{ "\n"}{.message}{ "\n"}'
```

**NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1 この例では、最新のリビジョン番号は 7 です。

出力に **2 nodes are at revision 6; 1 nodes are at revision 7** などの複数のリビジョン番号が含まれる場合、これは更新が依然として進行中であることを意味します。数分待機した後、再実行します。

- すべてのマスターホストが起動しており、クラスターに参加していることを確認します。クラスターにアクセスできるターミナルで、**cluster-admin** ユーザーとして以下のコマンドを実行します。

```
$ oc get pods -n openshift-etcd | grep etcd
```

### 出力例

```
etcd-ip-10-0-143-125.ec2.internal    2/2    Running    0      9h
etcd-ip-10-0-154-194.ec2.internal    2/2    Running    0      9h
etcd-ip-10-0-173-171.ec2.internal    2/2    Running    0      9h
```

この手順の完了後、すべてのサービスを復元するまでに数分かかる場合があります。たとえば、**oc login** を使用した認証は、OAuth サーバー Pod が再起動するまですぐに機能しない可能性があります。

## 1.9. POD の DISRUPTION BUDGET (停止状態の予算)

Pod の Disruption Budget について理解し、これを設定します。

### 1.9.1. Pod の Disruption Budget (停止状態の予算) を使って起動している Pod の数を指定する方法

Pod の Disruption Budget は [Kubernetes](#) API の一部であり、他のオブジェクトタイプのように **oc** コマンドで管理できます。この設定により、メンテナランスのためのノードのドレイン (解放) などの操作時に Pod への安全面の各種の制約を指定できます。

**PodDisruptionBudget** は、同時に起動している必要のあるレプリカの最小数またはパーセンテージを指定する API オブジェクトです。これらをプロジェクトに設定することは、ノードのメンテナランス (クラスターのスケールダウンまたはクラスターのアップグレードなどの実行) 時に役立ち、この設定は (ノードの障害時ではなく) 自発的なエビクションの場合にのみ許可されます。

**PodDisruptionBudget** オブジェクトの設定は、以下の主要な部分で設定されています。

- 一連の Pod に対するラベルのクエリー機能であるラベルセクター。
- 同時に利用可能にする必要のある Pod の最小数を指定する可用性レベル。
  - **minAvailable** は、中断時にも常に利用可能である必要のある Pod 数です。
  - **maxUnavailable** は、中断時に利用不可にできる Pod 数です。



#### 注記

**maxUnavailable** の **0%** または **0** あるいは **minAvailable** の **100%**、ないしはレプリカ数に等しい値は許可されますが、これによりノードがドレイン (解放) されないようにブロックされる可能性があります。

以下を実行して、Pod の Disruption Budget をすべてのプロジェクトで確認することができます。

```
$ oc get poddisruptionbudget --all-namespaces
```

#### 出力例

NAMESPACE	NAME	MIN-AVAILABLE	SELECTOR
another-project	another-pdb	4	bar=foo
test-project	my-pdb	2	foo=bar

**PodDisruptionBudget** は、最低でも **minAvailable** Pod がシステムで実行されている場合は正常であるとみなされます。この制限を超えるすべての Pod はエビクションの対象となります。



#### 注記

Pod の優先順位およびプリエンプシヨンの設定に基づいて、優先順位の低い Pod は Pod の Disruption Budget の要件を無視して削除される可能性があります。

### 1.9.2. Pod の Disruption Budget を使って起動している Pod 数の指定

同時に起動している必要のあるレプリカの最小数またはパーセンテージは、**PodDisruptionBudget** オブジェクトを使って指定します。

#### 手順

Pod の Disruption Budget を設定するには、以下を実行します。

1. YAML ファイルを以下のようなオブジェクト定義で作成します。

```
apiVersion: policy/v1beta1 ❶
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 ❷
  selector: ❸
    matchLabels:
      foo: bar
```

-

- ① **PodDisruptionBudget** は **policy/v1beta1** API グループの一部です。
- ② 同時に利用可能である必要のある Pod の最小数。これには、整数またはパーセンテージ (例: **20%**) を指定する文字列を使用できます。
- ③ 一連のリソースに対するラベルのクエリー。**matchLabels** と **matchExpressions** の結果は論理的に結合されます。

または、以下を実行します。

```
apiVersion: policy/v1beta1 ①
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% ②
  selector: ③
    matchLabels:
      foo: bar
```

- ① **PodDisruptionBudget** は **policy/v1beta1** API グループの一部です。
- ② 同時に利用不可にできる Pod の最大数。これには、整数またはパーセンテージ (例: **20%**) を指定する文字列を使用できます。
- ③ 一連のリソースに対するラベルのクエリー。**matchLabels** と **matchExpressions** の結果は論理的に結合されます。

2. 以下のコマンドを実行してオブジェクトをプロジェクトに追加します。

```
$ oc create -f </path/to/file> -n <project_name>
```

## 1.10. クラウドプロバイダーの認証情報のローテーションまたは削除

OpenShift Container Platform のインストール後に、一部の組織では、初回インストール時に使用されたクラウドプロバイダーの認証情報のローテーションまたは削除が必要になります。

クラスターが新規の認証情報を使用できるようにするには、[Cloud Credential Operator \(CCO\)](#) が使用するシークレットを更新して、クラウドプロバイダーの認証情報を管理できるようにする必要があります。

### 1.10.1. クラウドプロバイダーの認証情報の削除

OpenShift Container Platform クラスターを Amazon Web Services (AWS) にインストールしたら、クラスターの **kube-system** namespace から管理者レベルの認証情報シークレットを削除できます。管理者レベルの認証情報は、アップグレードなどの昇格されたパーミッションを必要とする変更時にのみ必要です。





## 注記

z-stream 以外のアップグレードの前に、認証情報のシークレットを管理者レベルの認証情報と共に元に戻す必要があります。認証情報が存在しない場合は、アップグレードがブロックされる可能性があります。


## 前提条件

- クラスターが、CCO からのクラウド認証情報の削除をサポートするプラットフォームにインストールされている。

## 手順

1. Web コンソールの **Administrator** パースペクティブで、**Workloads** → **Secrets** に移動します。
2. **Secrets** ページの表で、AWS の **aws-creds** ルートシークレットを見つけます。

プラットフォーム	シークレット名
AWS	<b>aws-creds</b>

3. シークレットと同じ行にある **Options** メニュー  をクリックし、**Delete Secret** を選択します。

## 1.11. 非接続クラスターのイメージストリームの設定

OpenShift Container Platform を非接続環境でインストールした後に、Cluster Samples Operator のイメージストリームおよび **must-gather** イメージストリームを設定します。

### 1.11.1. サポートデータを収集するためのクラスターの準備

ネットワークが制限された環境を使用するクラスターは、Red Hat サポート用のデバッグデータを収集するために、デフォルトの **must-gather** イメージをインポートする必要があります。must-gather イメージはデフォルトでインポートされず、ネットワークが制限された環境のクラスターは、リモートリポジトリから最新のイメージをプルするためにインターネットにアクセスできません。

## 手順

1. ミラーレジストリーの信頼される CA を Cluster Samples Operator 設定の一部としてクラスターのイメージ設定オブジェクトに追加していない場合は、以下の手順を実行します。

- a. クラスターのイメージ設定オブジェクトを作成します。

```
$ oc create configmap registry-config --from-file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

- b. クラスターのイメージ設定オブジェクトに、ミラーに必要な信頼される CA を追加します。

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
```

2. インストールペイロードからデフォルトの must-gather イメージをインポートします。

```
$ oc import-image is/must-gather -n openshift
```

**oc adm must-gather** コマンドの実行時に、以下の例のように **--image** フラグを使用し、ペイロードイメージを参照します。

```
$ oc adm must-gather --image=$(oc adm release info --image-for must-gather)
```

## 関連情報

- [Amazon Web Services \(AWS\) シークレット形式](#)
- [Microsoft Azure シークレットの形式](#)
- [Google Cloud Platform \(GCP\) シークレット形式](#)

## 第2章 インストール後のノードタスク

OpenShift Container Platform のインストール後に、特定のノードタスクでクラスターをさらに拡張し、要件に合わせてカスタマイズできます。

### 2.1. RHEL コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加

RHEL コンピュータノードについて理解し、これを使用します。

#### 2.1.1. RHEL コンピュータノードのクラスターへの追加について

OpenShift Container Platform 4.5 には、ユーザーによってプロビジョニングされるインフラストラクチャを使用する場合、Red Hat Enterprise Linux (RHEL) マシンをクラスター内のコンピュータまたはワーカーマシンとして使用するオプションがあります。クラスター内のコントロールプレーンまたはマスターマシンには Red Hat Enterprise Linux CoreOS (RHCOS) マシンを使用する必要があります。

ユーザーによってプロビジョニングされるインフラストラクチャを使用するすべてのインストールの場合、クラスターで RHEL コンピュータマシンを使用する選択をする場合には、システム更新の実行や、パッチの適用、またその他の必要なすべてのタスクの実行を含むオペレーティングシステムのライフサイクル管理およびメンテナンスのすべてを独自に実行する必要があります。



#### 重要

OpenShift Container Platform をクラスター内のマシンから削除するには、オペレーティングシステムを破棄する必要があるため、クラスターに追加する RHEL マシンについては専用のハードウェアを使用する必要があります。



#### 重要

swap メモリーは、OpenShift Container Platform クラスターに追加されるすべての RHEL マシンで無効にされます。これらのマシンで swap メモリーを有効にすることはできません。

RHEL コンピュータマシンは、コントロールプレーンを初期化してからクラスターに追加する必要があります。

#### 2.1.2. RHEL コンピュータノードのシステム要件

OpenShift Container Platform 環境の Red Hat Enterprise Linux (RHEL) コンピュータマシンホスト (またはワーカーマシンホストとしても知られる) は以下の最低のハードウェア仕様およびシステムレベルの要件を満たしている必要があります。

- まず、お使いの Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションがなければなりません。これがない場合は、営業担当者にお問い合わせください。
- 実稼働環境では予想されるワークロードに対応するコンピューターノードを提供する必要があります。クラスター管理者は、予想されるワークロードを計算し、オーバーヘッドの約 10 パーセントを追加する必要があります。実稼働環境の場合、ノードホストの障害が最大容量に影響を与えることがないように、十分なリソースを割り当てるようにします。
- 各システムは、以下のハードウェア要件を満たしている必要があります。
  - 物理または仮想システム、またはパブリックまたはプライベート IaaS で実行されるインス

タンス。

- ベース OS: [RHEL 7.7-7.8](#) (最小のインストールオプション)



### 重要

OpenShift Container Platform 4.5 でサポートされるのは RHEL 7.7-7.8 のみになります。コンピュータマシンを RHEL 8 にアップグレードすることはできません。

- FIPS モードで OpenShift Container Platform をデプロイしている場合、起動する前に FIPS を RHEL マシン上で有効にする必要があります。詳細は、RHEL 7 のドキュメントの [FIPS モードの有効化](#) を参照してください。
- NetworkManager 1.0 以降。
- 1 vCPU。
- 最小 8 GB の RAM。
- `/var/` を含むファイルシステムの最小 15 GB のハードディスク領域。
- `/usr/local/bin/` を含むファイルシステムの最小 1 GB のハードディスク領域。
- システムの一時ディレクトリーを含むファイルシステムの最小 1 GB のハードディスク領域。システムの一時的ディレクトリーは、Python の標準ライブラリーの `tempfile` モジュールで定義されるルールに基づいて決定されます。
- 各システムは、システムプロバイダーの追加の要件を満たす必要があります。たとえば、クラスターを VMware vSphere にインストールしている場合、ディスクはその [ストレージガイドライン](#) に応じて設定され、`disk.enableUUID=true` 属性が設定される必要があります。
- 各システムは、DNS で解決可能なホスト名を使用してクラスターの API エンドポイントにアクセスする必要があります。配置されているネットワークセキュリティアクセス制御は、クラスターの API サービスエンドポイントへのシステムアクセスを許可する必要があります。

#### 2.1.2.1. 証明書署名要求の管理

ユーザーがプロビジョニングするインフラストラクチャーを使用する場合、クラスターの自動マシン管理へのアクセスは制限されるため、インストール後にクラスターの証明書署名要求 (CSR) のメカニズムを提供する必要があります。**kube-controller-manager** は kubelet クライアント CSR のみを承認します。**machine-approver** は、kubelet 認証情報を使用して要求される提供証明書の有効性を保証できません。適切なマシンがこの要求を発行したかどうかを確認できないためです。kubelet 提供証明書の要求の有効性を検証し、それらを承認する方法を判別し、実装する必要があります。

#### 2.1.3. Playbook 実行のためのマシンの準備

Red Hat Enterprise Linux をオペレーティングシステムとして使用するコンピュータマシンを OpenShift Container Platform 4.5 クラスターに追加する前に、Playbook を実行するマシンを準備する必要があります。このマシンはクラスターの一部にはなりませんが、クラスターにアクセスする必要があります。

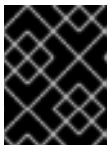
#### 前提条件

- Playbook を実行するマシンに OpenShift CLI (**oc**) 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ログインします。

## 手順

1. クラスターの **kubeconfig** ファイルおよびクラスターのインストールに使用したインストールプログラムがマシン上にあることを確認します。これを実行する1つの方法として、クラスターのインストールに使用したマシンと同じマシンを使用することができます。
2. マシンを、コンピュータマシンとして使用する予定のすべての RHEL ホストにアクセスできるように設定します。Bastion と SSH プロキシまたは VPN の使用など、所属する会社で許可されるすべての方法を利用できます。
3. すべての RHEL ホストへの SSH アクセスを持つユーザーを Playbook を実行するマシンで設定します。



### 重要

SSH キーベースの認証を使用する場合、キーを SSH エージェントで管理する必要があります。

4. これを実行していない場合には、マシンを RHSM に登録し、**OpenShift** サブスクリプションのプールをこれにアタッチします。

- a. マシンを RHSM に登録します。

```
# subscription-manager register --username=<user_name> --password=<password>
```

- b. RHSM から最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

- c. 利用可能なサブスクリプションを一覧表示します。

```
# subscription-manager list --available --matches '*OpenShift*'
```

- d. 直前のコマンドの出力で、OpenShift Container Platform サブスクリプションのプール ID を見つけ、これをアタッチします。

```
# subscription-manager attach --pool=<pool_id>
```

5. OpenShift Container Platform 4.5 で必要なりポジトリを有効にします。

```
# subscription-manager repos \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-server-extras-rpms" \
  --enable="rhel-7-server-ansible-2.9-rpms" \
  --enable="rhel-7-server-ose-4.5-rpms"
```

6. **openshift-ansible** を含む必要なパッケージをインストールします。

```
# yum install openshift-ansible openshift-clients jq
```

**openshift-ansible** パッケージはインストールプログラムユーティリティを提供し、Ansible

Playbook などのクラスターに RHEL コンピュートノードを追加するために必要な他のパッケージおよび関連する設定ファイルをプルします。**openshift-clients** は **oc** CLI を提供し、**jq** パッケージはコマンドライン上での JSON 出力の表示方法を向上させます。

## 2.1.4. RHEL コンピュートノードの準備

Red Hat Enterprise Linux (RHEL) マシンを OpenShift Container Platform クラスターに追加する前に、各ホストを Red Hat Subscription Manager (RHSM) に登録し、有効な OpenShift Container Platform サブスクリプションをアタッチし、必要なりポジトリを有効にする必要があります。

1. 各ホストで RHSM に登録します。

```
# subscription-manager register --username=<user_name> --password=<password>
```

2. RHSM から最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

3. 利用可能なサブスクリプションを一覧表示します。

```
# subscription-manager list --available --matches '*OpenShift*'
```

4. 直前のコマンドの出力で、OpenShift Container Platform サブスクリプションのプール ID を見つけ、これをアタッチします。

```
# subscription-manager attach --pool=<pool_id>
```

5. yum リポジトリをすべて無効にします。

- a. 有効にされている RHSM リポジトリをすべて無効にします。

```
# subscription-manager repos --disable='*'
```

- b. 残りの yum リポジトリを一覧表示し、**repo id** にあるそれらの名前をメモします (ある場合)。

```
# yum repolist
```

- c. **yum-config-manager** を使用して、残りの yum リポジトリを無効にします。

```
# yum-config-manager --disable <repo_id>
```

または、すべてのリポジトリを無効にします。

```
# yum-config-manager --disable \*
```

利用可能なリポジトリが多い場合には、数分の時間がかかることがあります。

6. OpenShift Container Platform 4.5 で必要なりポジトリのみを有効にします。

```
# subscription-manager repos \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-server-extras-rpms" \
```

```
--enable="rhel-7-server-ose-4.5-rpms"
```

7. ホストで firewalld を停止し、無効にします。

```
# systemctl disable --now firewalld.service
```



### 注記

firewalld は、後で有効にすることはできません。これを実行する場合、ワーカ上の OpenShift Container Platform ログにはアクセスできません。

## 2.1.5. RHEL コンピュータマシンのクラスターへの追加

Red Hat Enterprise Linux をオペレーティングシステムとして使用するコンピュータマシンを OpenShift Container Platform 4.5 クラスターに追加することができます。

### 前提条件

- Playbook を実行するマシンに必要なパッケージをインストールし、必要な設定が行われています。
- インストール用の RHEL ホストを準備しています。

### 手順

Playbook を実行するために準備しているマシンで以下の手順を実行します。

1. コンピュータマシンホストおよび必要な変数を定義する `<path>/inventory/hosts` という名前の Ansible インベントリーファイルを作成します。

```
[all:vars]
ansible_user=root ❶
#ansible_become=True ❷

openshift_kubeconfig_path=~/.kube/config" ❸

[new_workers] ❹
mycluster-rhel7-0.example.com
mycluster-rhel7-1.example.com
```

- ❶ Ansible タスクをリモートコンピュータマシンで実行するユーザー名を指定します。
- ❷ **ansible\_user** の **root** を指定しない場合、**ansible\_become** を **True** に設定し、ユーザーに sudo パーミッションを割り当てる必要があります。
- ❸ クラスターの **kubeconfig** ファイルへのパスを指定します。
- ❹ クラスターに追加する各 RHEL マシンを一覧表示します。各ホストについて完全修飾ドメイン名を指定する必要があります。この名前は、クラスターがマシンにアクセスするために使用するホスト名であるため、マシンにアクセスできるように正しいパブリックまたはプライベートの名前を設定します。

2. Ansible Playbook ディレクトリーに移動します。

■

```
$ cd /usr/share/ansible/openshift-ansible
```

3. Playbook を実行します。

```
$ ansible-playbook -i /<path>/inventory/hosts playbooks/scaleup.yml 1
```

- 1** **<path>** については、作成した Ansible インベントリーファイルへのパスを指定します。

## 2.1.6. Ansible ホストファイルの必須パラメーター

Red Hat Enterprise Linux (RHEL) コンピュータマシンをクラスターに追加する前に、以下のパラメーターを Ansible ホストファイルに定義する必要があります。

パラメーター	説明	値
<b>ansible_user</b>	パスワードなしの SSH ベースの認証を許可する SSH ユーザー。SSH キーベースの認証を使用する場合、キーを SSH エージェントで管理する必要があります。	システム上のユーザー名。デフォルト値は <b>root</b> です。
<b>ansible_become</b>	<b>ansible_user</b> の値が root ではない場合、 <b>ansible_become</b> を <b>True</b> に設定する必要があります、 <b>ansible_user</b> として指定するユーザーはパスワードなしの sudo アクセスが可能になるように設定される必要があります。	<b>True</b> 。値が <b>True</b> ではない場合、このパラメーターを指定したり、定義したりしないでください。
<b>openshift_kubeconfig_path</b>	クラスターの <b>kubeconfig</b> ファイルが含まれるローカルディレクトリーへのパスおよびファイル名を指定します。	設定ファイルのパスと名前。

## 2.1.7. オプション: RHCOS コンピュータマシンのクラスターからの削除

Red Hat Enterprise Linux (RHEL) コンピュータマシンをクラスターに追加した後に、オプションで Red Hat Enterprise Linux CoreOS (RHCOS) コンピュータマシンを削除し、リソースを解放できます。

### 前提条件

- RHEL コンピュータマシンをクラスターに追加済みです。

### 手順

1. マシンの一覧を表示し、RHCOS コンピュータマシンのノード名を記録します。

```
$ oc get nodes -o wide
```

2. それぞれの RHCOS コンピュータマシンについて、ノードを削除します。
  - a. **oc adm cordon** コマンドを実行して、ノードにスケジューリング対象外 (unschedulable) のマークを付けます。



```
$ oc adm cordon <node_name> ❶
```

- ❶ RHCOS コンピュータマシンのノード名を指定します。

- b. ノードからすべての Pod をドレイン (解放) します。

```
$ oc adm drain <node_name> --force --delete-local-data --ignore-daemonsets ❶
```

- ❶ 分離した RHCOS コンピュータマシンのノード名を指定します。

- c. ノードを削除します。

```
$ oc delete nodes <node_name> ❶
```

- ❶ ドレイン (解放) した RHCOS コンピュータマシンのノード名を指定します。

3. コンピュータマシンの一覧を確認し、RHEL ノードのみが残っていることを確認します。

```
$ oc get nodes -o wide
```

4. RHCOS マシンをクラスターのコンピュータマシンのロードバランサーから削除します。仮想マシンを削除したり、RHCOS コンピュータマシンの物理ハードウェアを再イメージ化したりできます。

## 2.2. RHCOS コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加

ベアメタルの OpenShift Container Platform クラスターに Red Hat Enterprise Linux CoreOS (RHCOS) コンピュータマシンを追加することができます。

ベアメタルインフラストラクチャーにインストールされているクラスターにコンピュータマシンを追加する前に、それが使用する RHCOS マシンを作成する必要があります。ISO イメージまたはネットワーク PXE ブートを使用してマシンを作成できます。

### 2.2.1. 前提条件

- クラスターをベアメタルにインストールしている。
- クラスターの作成に使用したインストールメディアおよび Red Hat Enterprise Linux CoreOS (RHCOS) イメージがある。これらのファイルがない場合は、[インストール手順](#)に従ってこれらを取得する必要があります。

### 2.2.2. ISO イメージを使用した追加の RHCOS マシンの作成

ISO イメージを使用して、ベアメタルクラスターの追加の Red Hat Enterprise Linux CoreOS (RHCOS) コンピュータマシンを作成できます。

#### 前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- クラスターのインストール時に HTTP サーバーにアップロードした BIOS または UEFI RHCOS イメージファイルの URL を取得します。

## 手順

1. ISO ファイルを使用して、追加のコンピュートマシンに RHCOS をインストールします。クラスターのインストール前にマシンを作成する際に使用したのと同じ方法を使用します。
  - ディスクに ISO イメージを書き込み、これを直接起動します。
  - LOM インターフェイスで ISO リダイレクトを使用します。
2. インスタンスの起動後に、**TAB** または **E** キーを押してカーネルコマンドラインを編集します。
3. パラメーターをカーネルコマンドラインに追加します。

```
coreos.inst=yes
coreos.inst.install_dev=sda ❶
coreos.inst.image_url=<bare_metal_image_URL> ❷
coreos.inst.ignition_url=http://example.com/worker.ign ❸
```

- ❶ インストール先のシステムのブロックデバイスを指定します。
  - ❷ サーバーにアップロードした UEFI または BIOS イメージの URL を指定します。
  - ❸ コンピュート Ignition 設定ファイルの URL を指定します。
4. **Enter** を押してインストールを完了します。RHCOS のインストール後に、システムは再起動します。システムの再起動後、指定した Ignition 設定ファイルを適用します。
  5. 継続してクラスター用の追加のコンピュートマシンを作成します。

### 2.2.3. PXE または iPXE ブートによる追加の RHCOS マシンの作成

PXE または iPXE ブートを使用して、ベアメタルクラスターの追加の Red Hat Enterprise Linux CoreOS (RHCOS) コンピュートマシンを作成できます。

#### 前提条件

- クラスターのコンピュートマシンの Ignition 設定ファイルの URL を取得します。このファイルがインストール時に HTTP サーバーにアップロードされている必要があります。
- クラスターのインストール時に HTTP サーバーにアップロードした RHCOS ISO イメージ、圧縮されたメタル BIOS、**kernel**、および **initramfs** ファイルの URL を取得します。
- インストール時に OpenShift Container Platform クラスターのマシンを作成するために使用した PXE ブートインフラストラクチャーにアクセスする必要があります。RHCOS のインストール後にマシンはローカルディスクから起動する必要があります。
- UEFI を使用する場合、OpenShift Container Platform のインストール時に変更した **grub.conf** ファイルにアクセスできます。

## 手順

1. RHCOS イメージの PXE または iPXE インストールが正常に行われていることを確認します。

- PXE の場合:

```

DEFAULT pxeboot
TIMEOUT 20
PROMPT 0
LABEL pxeboot
  KERNEL http://<HTTP_server>/rhcos-<version>-installer-kernel-<architecture> ❶
  APPEND ip=dhcp rd.neednet=1 initrd=http://<HTTP_server>/rhcos-<version>-installer-
initramfs.<architecture>.img coreos.inst=yes coreos.inst.install_dev=sda
coreos.inst.image_url=http://<HTTP_server>/rhcos-<version>-metal.
<architecture>.raw.gz coreos.inst.ignition_url=http://<HTTP_server>/worker.ign ❷ ❸

```

- ❶ HTTP サーバーにアップロードした **kernel** ファイルの場所を指定します。
- ❷ 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。
- ❸ HTTP サーバーにアップロードした RHCOS ファイルの場所を指定します。**initrd** パラメーター値は **initramfs** ファイルの場所であり、**coreos.inst.image\_url** パラメーター値は圧縮された metal RAW イメージの場所、および **coreos.inst.ignition\_url** パラメーター値はワーカー Ignition 設定ファイルの場所になります。



## 注記

この設定では、グラフィカルコンソールを使用するマシンでシリアルコンソールアクセスを有効にしません。別のコンソールを設定するには、**APPEND** 行に1つ以上の **console=** 引数を追加します。たとえば、**console=tty0 console=ttyS0** を追加して、最初の PC シリアルポートをプライマリーコンソールとして、グラフィカルコンソールをセカンダリーコンソールとして設定します。詳細は、[How does one set up a serial terminal and/or console in Red Hat Enterprise Linux?](#) を参照してください。

- iPXE の場合:

```

kernel http://<HTTP_server>/rhcos-<version>-installer-kernel-<architecture> ip=dhcp
rd.neednet=1 initrd=http://<HTTP_server>/rhcos-<version>-installer-initramfs.
<architecture>.img coreos.inst=yes coreos.inst.install_dev=sda
coreos.inst.image_url=http://<HTTP_server>/rhcos-<version>-metal.
<architecture>.raw.gz coreos.inst.ignition_url=http://<HTTP_server>/worker.ign ❶ ❷
initrd http://<HTTP_server>/rhcos-<version>-installer-initramfs.<architecture>.img ❸
boot

```

- ❶ HTTP サーバーにアップロードした RHCOS ファイルの場所を指定します。**kernel** パラメーター値は **kernel** ファイルの場所であり、**initrd** パラメーター値は **initramfs** ファイルの場所、**coreos.inst.image\_url** パラメーター値は圧縮された metal RAW イメージの場所、および **coreos.inst.ignition\_url** パラメーター値はワーカー Ignition 設定ファイルの場所になります。
- ❷ 複数の NIC を使用する場合、**ip** オプションに単一インターフェイスを指定します。たとえば、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定し

これは、**eno1** という名前の NIC で DHCP を使用するには、**ip=eno1:dhcp** を設定します。

- 3 HTTP サーバーにアップロードした **initramfs** ファイルの場所を指定します。



#### 注記

この設定では、グラフィカルコンソールを使用するマシンでシリアルコンソールアクセスを有効にしません。別のコンソールを設定するには、**kernel** 行に **console=** 引数を1つ以上追加します。たとえば、**console=tty0 console=ttyS0** を追加して、最初の PC シリアルポートをプライマリコンソールとして、グラフィカルコンソールをセカンダリーコンソールとして設定します。詳細は、[How does one set up a serial terminal and/or console in Red Hat Enterprise Linux?](#) を参照してください。

2. PXE または iPXE インフラストラクチャーを使用して、クラスターに必要なコンピュータマシンを作成します。

### 2.2.4. マシンの証明書署名要求の承認

マシンをクラスターに追加する際に、追加したそれぞれのマシンについて2つの保留状態の証明書署名要求 (CSR) が生成されます。これらの CSR が承認されていることを確認するか、または必要な場合はそれらを承認してください。最初にクライアント要求を承認し、次にサーバー要求を承認する必要があります。

#### 前提条件

- マシンがクラスターに追加されています。

#### 手順

1. クラスターがマシンを認識していることを確認します。

```
$ oc get nodes
```

#### 出力例

```
NAME      STATUS    ROLES    AGE   VERSION
master-0  Ready     master   63m   v1.18.3
master-1  Ready     master   63m   v1.18.3
master-2  Ready     master   64m   v1.18.3
worker-0  NotReady  worker   76s   v1.18.3
worker-1  NotReady  worker   70s   v1.18.3
```

出力には作成したすべてのマシンが一覧表示されます。

2. 保留中の証明書署名要求 (CSR) を確認し、クラスターに追加したそれぞれのマシンのクライアントおよびサーバー要求に **Pending** または **Approved** ステータスが表示されていることを確認します。

```
$ oc get csr
```

#### 出力例

NAME	AGE	REQUESTOR	CONDITION
csr-8b2br	15m	system:serviceaccount:openshift-machine-config-operator:node-bootstrap	Pending
csr-8vnps	15m	system:serviceaccount:openshift-machine-config-operator:node-bootstrap	Pending
...			

この例では、2つのマシンがクラスターに参加しています。この一覧にはさらに多くの承認された CSR が表示される可能性があります。

- 追加したマシンの保留中の CSR すべてが **Pending** ステータスになった後に CSR が承認されない場合には、クラスターマシンの CSR を承認します。



### 注記

CSR のローテーションは自動的に実行されるため、クラスターにマシンを追加後 1 時間以内に CSR を承認してください。1 時間以内に承認しない場合には、証明書のローテーションが行われ、各ノードに 3 つ以上の証明書が存在するようになります。これらの証明書すべてを承認する必要があります。クライアントの CSR が承認されたら、Kubelet は提供証明書のセカンダリー CSR を作成します。これには、手動の承認が必要です。次に、後続の提供証明書の更新要求は、Kubelet が同じパラメーターを持つ新規証明書を要求する場合に **machine-approver** によって自動的に承認されます。

- それらを個別に承認するには、それぞれの有効な CSR について以下のコマンドを実行します。

```
$ oc adm certificate approve <csr_name> ❶
```

❶ **<csr\_name>** は、現行の CSR の一覧からの CSR の名前です。

- すべての保留中の CSR を承認するには、以下のコマンドを実行します。

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```

- クライアント要求が承認されたら、クラスターに追加した各マシンのサーバー要求を確認する必要があります。

```
$ oc get csr
```

### 出力例

NAME	AGE	REQUESTOR	CONDITION
csr-bfd72	5m26s	system:node:ip-10-0-50-126.us-east-2.compute.internal	Pending
csr-c57lv	5m26s	system:node:ip-10-0-95-157.us-east-2.compute.internal	Pending
...			

- 残りの CSR が承認されず、それらが **Pending** ステータスにある場合、クラスターマシンの CSR を承認します。

- それらを個別に承認するには、それぞれの有効な CSR について以下のコマンドを実行します。

```
$ oc adm certificate approve <csr_name> ❶
```

❶ **<csr\_name>** は、現行の CSR の一覧からの CSR の名前です。

- すべての保留中の CSR を承認するには、以下のコマンドを実行します。

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}{{end}}' | xargs oc adm certificate approve
```

- すべてのクライアントおよびサーバーの CSR が承認された後に、マシンのステータスが **Ready** になります。以下のコマンドを実行して、これを確認します。

```
$ oc get nodes
```

### 出力例

```
NAME      STATUS   ROLES    AGE   VERSION
master-0  Ready    master   73m   v1.20.0
master-1  Ready    master   73m   v1.20.0
master-2  Ready    master   74m   v1.20.0
worker-0  Ready    worker   11m   v1.20.0
worker-1  Ready    worker   11m   v1.20.0
```



### 注記

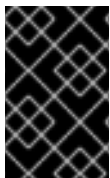
サーバー CSR の承認後にマシンが **Ready** ステータスに移行するまでに数分の時間がかかる場合があります。

### 関連情報

- CSR の詳細は、[Certificate Signing Requests](#) を参照してください。

## 2.3. マシンヘルスチェックのデプロイ

マシンヘルスチェックについて確認し、これをデプロイします。



### 重要

このプロセスは、マシンを独自に手動でプロビジョニングしているクラスターには適用されません。高度なマシン管理およびスケーリング機能は、マシン API が機能しているクラスターでのみ使用することができます。

### 2.3.1. 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。

#### 2.3.1.1. 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。

#### 2.3.1.2. 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。



### 2.3.2. 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 ❶
  namespace: openshift-machine-api
  annotations:
    machine.openshift.io/remediation-strategy: external-baremetal ❷
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> ❸
      machine.openshift.io/cluster-api-machine-type: <role> ❹
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> ❺
  unhealthyConditions:
    - type: "Ready"
      timeout: "300s" ❻
      status: "False"
    - type: "Ready"
      timeout: "300s" ❼
      status: "Unknown"
  maxUnhealthy: "40%" ❽
  nodeStartupTimeout: "10m" ❾
```

- ❶ 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。
- ❷ 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。
- ❸ ❹ 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。
- ❺ 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。
- ❻ ❼ 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。
- ❽ 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。
- ❾ 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が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 ❶
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> ❷
      machine.openshift.io/cluster-api-machine-type: <role> ❸
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> ❹
  unhealthyConditions:
    - type: "Ready"
      timeout: "300s" ❺
      status: "False"
    - type: "Ready"
      timeout: "300s" ❻
      status: "Unknown"
  maxUnhealthy: "40%" ❼
  nodeStartupTimeout: "10m" ❽
```

- ❶ 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。
- ❷ ❸ 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。
- ❹ 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。
- ❺ ❻ 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。
- ❼ 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。
- ❽ 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。

#### 2.3.2.1. 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。

#### 2.3.2.1.1. 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。

#### 2.3.2.1.2. 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。

#### 2.3.3. MachineHealthCheck リソースの作成

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

#### 前提条件

- **oc** コマンドラインインターフェイスをインストールします。

#### 手順

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

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

### 2.3.4. マシンセットの手動によるスケーリング

マシンセットのマシンのインスタンスを追加したり、削除したりする必要がある場合、マシンセットを手動でスケーリングできます。

本書のガイダンスは、完全に自動化された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のインストールに関連します。ユーザーによってプロビジョニングされるインフラストラクチャのカスタマイズされたインストールにはマシンセットがありません。

#### 前提条件

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

#### 手順

1. クラスターにあるマシンセットを表示します。

```
$ oc get machinesets -n openshift-machine-api
```

マシンセットは **<clusterid>-worker-<aws-region-az>** の形式で一覧表示されます。

2. マシンセットをスケーリングします。

```
$ 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
```

マシンセットをスケールアップまたはスケールダウンできます。新規マシンが利用可能になるまで数分の時間がかかります。

### 2.3.5. マシンセットとマシン設定プールの相違点について

**MachineSet** オブジェクトは、クラウドまたはマシンプロバイダーに関する OpenShift Container Platform ノードを記述します。

**MachineConfigPool** オブジェクトにより、**MachineConfigController** コンポーネントがアップグレードのコンテキストでマシンのステータスを定義し、提供できるようになります。

**MachineConfigPool** オブジェクトにより、ユーザーはマシン設定プールの OpenShift Container Platform ノードにアップグレードを展開する方法を設定できます。

**NodeSelector** オブジェクトは **MachineSet** オブジェクト への参照に置き換えることができます。

## 2.4. 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、**podsWithCore** 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、単一コンテナを保持する Pod は実際には 2 つのコンテナを使用します。2 つ目のコンテナは実際のコンテナ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 のコンテナが実行されていることになります。

**podsWithCore** は、ノードのプロセッサ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 プロセッサコアを搭載したノードで **podsWithCore** 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:
  podsWithCore: 10
```

**podsWithCore** 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。**podsWithCore** 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
```

### 2.4.1. 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

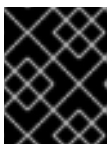
デフォルトでは、kubelet 関連の設定を利用可能なワーカーノードに適用する場合に1つのマシンのみを利用不可の状態にすることが許可されます。大規模なクラスターの場合、設定の変更が反映されるまでに長い時間がかかる可能性があります。プロセスのスピードを上げるためにマシン数の調整をいつでも実行することができます。

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。

2.4.2. コントロールプレーンノードのサイジング

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

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

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

3 つのコントロールプレーンノード (またはマスターノード) 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最大で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た場合、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。

## 2.4.3. 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 ❶
  cpuManagerReconcilePeriod: 5s ❷
```

### ❶ 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。

### ❷ 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 ❶
cpuManagerReconcilePeriod: 5s ❷
```

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

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

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

### 出力例

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

9. Pod を作成します。

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

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

```
# oc describe pod cpumanager
```

### 出力例

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

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

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

QoS (quality of service) 階層 **Guaranteed** の Pod は、**kubepods.slice** に配置されます。他の QoS の Pod は、**kubepods** の子である **cgrouops** 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
```

- 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
```

- 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

## 2.5. HUGE PAGE

Huge Page について理解し、これを設定します。

### 2.5.1. 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 (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せんが、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。

### 2.5.2. 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、これはスケジューラ可能なリソース **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 ❶
      memory: "1Gi"
      cpu: "1"
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

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

### 指定されたサイズの Huge Page の割り当て

プラットフォームによっては、複数の Huge Page サイズをサポートするものもあります。特定のサイズの Huge Page を割り当てるには、Huge Page の起動コマンドパラメーターの前に、Huge Page サイズの選択パラメーター **hugepagesz=<size>** を指定してください。**<size>** の値は、バイトで指定する必要があります。その際、オプションでスケール接尾辞 [**kKmMgG**] を指定できます。デフォルトの Huge Page サイズは、**default\_hugepagesz=<size>** の起動パラメーターで定義できます。

### Huge page の要件

- Huge Page 要求は制限と同じでなければなりません。制限が指定されているにもかかわらず、要求が指定されていない場合には、これがデフォルトになります。

- Huge Page は、Pod のスコープで分割されます。コンテナの分割は、今後のバージョンで予定されています。
- Huge Page がサポートする **EmptyDir** ボリュームは、Pod 要求よりも多くの Huge Page メモリーを消費することはできません。
- **shmget()** で **SHM\_HUGETLB** を使用して Huge Page を消費するアプリケーションは、**proc/sys/vm/hugetlb\_shm\_group** に一致する補助グループで実行する必要があります。

## 関連情報

- [Transparent Huge Page の設定](#)

## 2.5.3. 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。

### 2.5.3.1. 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 ❶
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: ❷
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 ❸
      name: openshift-node-hugepages

  recommend:
    - machineConfigLabels: ❹
        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。

## 2.6. デバイスプラグインについて

デバイスプラグインは、クラスター間でハードウェアデバイスを使用する際の一貫した移植可能なソリューションを提供します。デバイスプラグインは、拡張メカニズムを通じてこれらのデバイスをサポートし(これにより、コンテナーがこれらのデバイスを利用できるようになります)、デバイスのヘルスチェックを実施し、それらを安全に共有します。



## 重要

OpenShift Container Platform はデバイスのプラグイン API をサポートしますが、デバイスプラグインコンテナーは個別のベンダーによりサポートされます。

デバイスプラグインは、特定のハードウェアリソースの管理を行う、ノード上で実行される gRPC サービスです (**kubelet** の外部にあります)。デバイスプラグインは以下のリモートプロシーチャーコール (RPC) をサポートしている必要があります。

```
service DevicePlugin {
  // GetDevicePluginOptions returns options to be communicated with Device
  // Manager
  rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

  // ListAndWatch returns a stream of List of Devices
  // Whenever a Device state change or a Device disappears, ListAndWatch
  // returns the new list
  rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

  // Allocate is called during container creation so that the Device
  // Plug-in can run device specific operations and instruct Kubelet
  // of the steps to make the Device available in the container
  rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

  // PreStartcontainer is called, if indicated by Device Plug-in during
  // registration phase, before each container start. Device plug-in
  // can run device specific operations such as resetting the device
  // before making devices available to the container
  rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

## デバイスプラグインの例

- [COS ベースのオペレーティングシステム用の Nvidia GPU デバイスプラグイン](#)
- [Nvidia の公式 GPU デバイスプラグイン](#)
- [Solarflare デバイスプラグイン](#)
- [KubeVirt デバイスプラグイン: vfio および kvm](#)



## 注記

デバイスプラグイン参照の実装を容易にするため、[vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device\\_plugin\\_stub.go](#) という Device Manager コードのスタブデバイスプラグインを使用できます。

### 2.6.1. デバイスプラグインのデプロイ方法

- デーモンセットは、デバイスプラグインのデプロイメントに推奨される方法です。

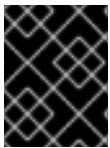


- 起動時にデバイスプラグインは、デバイスマネージャーから RPC を送信するためにノードの `/var/lib/kubelet/device-plugin/` での UNIX ドメインソケットの作成を試行します。
- デバイスプラグインは、ソケットの作成のほかにもハードウェアリソース、ホストファイルシステムへのアクセスを管理する必要があるため、特権付きセキュリティコンテキストで実行される必要があります。
- デプロイメント手順の詳細については、それぞれのデバイスプラグインの実装で確認できます。

### 2.6.2. デバイスマネージャーについて

デバイスマネージャーは、特殊なノードのハードウェアリソースを、デバイスプラグインとして知られるプラグインを使って公開するメカニズムを提供します。

特殊なハードウェアは、アップストリームのコード変更なしに公開できます。



#### 重要

OpenShift Container Platform はデバイスのプラグイン API をサポートしますが、デバイスプラグインコンテナは個別のベンダーによりサポートされます。

デバイスマネージャーはデバイスを **拡張リソース** として公開します。ユーザー Pod は、他の **拡張リソース** を要求するために使用されるのと同じ **制限/要求** メカニズムを使用してデバイスマネージャーで公開されるデバイスを消費できます。

使用開始時に、デバイスプラグインは `/var/lib/kubelet/device-plugins/kubelet.sock` の **Register** を起動してデバイスマネージャーに自己登録し、デバイスマネージャーの要求を提供するために `/var/lib/kubelet/device-plugins/<plugin>.sock` で gRPC サービスを起動します。

デバイスマネージャーは、新規登録要求の処理時にデバイスプラグインサービスで **ListAndWatch** リモートプロシージャーコール (RPC) を起動します。応答としてデバイスマネージャーは gRPC ストリームでプラグインから **デバイス** オブジェクトの一覧を取得します。デバイスマネージャーはプラグインからの新規の更新の有無についてストリームを監視します。プラグイン側では、プラグインはストリームを開いた状態にし、デバイスの状態に変更があった場合には常に新規デバイスの一覧が同じストリーム接続でデバイスマネージャーに送信されます。

新規 Pod の受付要求の処理時に、Kubelet はデバイスの割り当てのために要求された **Extended Resource** をデバイスマネージャーに送信します。デバイスマネージャーはそのデータベースにチェックインして対応するプラグインが存在するかどうかを確認します。プラグインが存在し、ローカルキャッシュと共に割り当て可能な空きデバイスがある場合、**Allocate** RPC がその特定デバイスのプラグインで起動します。

さらにデバイスプラグインは、ドライバーのインストール、デバイスの初期化、およびデバイスのリセットなどの他のいくつかのデバイス固有の操作も実行できます。これらの機能は実装ごとに異なります。

### 2.6.3. デバイスマネージャーの有効化

デバイスマネージャーを有効にし、デバイスプラグインを実装してアップストリームのコード変更なしに特殊なハードウェアを公開できるようにします。

デバイスマネージャーは、特殊なノードのハードウェアリソースを、デバイスプラグインとして知られるプラグインを使って公開するメカニズムを提供します。

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

```
# oc describe machineconfig <name>
```

以下に例を示します。

```
# oc describe machineconfig 00-worker
```

### 出力例

```
Name:      00-worker
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker 1
```

- 1** デバイスマネージャーに必要なラベル。

## 手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

### Device Manager CR の設定例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr 2
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true 3
```

- 1** CR に名前を割り当てます。
- 2** Machine Config Pool からラベルを入力します。
- 3** **DevicePlugins** を 'true' に設定します。

2. デバイスマネージャーを作成します。

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

### 出力例

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. デバイスマネージャーが実際に有効にされるように、`/var/lib/kubelet/device-plugins/kubelet.sock` がノードで作成されていることを確認します。これは、デバイスマネージャーの gRPC サーバーが新規プラグインの登録がないかどうかリッスンする UNIX ドメインソケットです。このソケットファイルは、デバイスマネージャーが有効にされている場合にのみ Kubelet の起動時に作成されます。

## 2.7. テイントおよび容認 (TOLERATION)

テイントおよび容認について理解し、これらを使用します。

### 2.7.1. テイントおよび容認 (Toleration) について

テイントにより、ノードは Pod に一致する **容認** がない場合に Pod のスケジュールを拒否することができます。

テイントは **Node** 仕様 (**NodeSpec**) でノードに適用され、容認は **Pod** 仕様 (**PodSpec**) で Pod に適用されます。テイントをノードに適用する場合、スケジューラーは Pod がテイントを容認しない限り、Pod をそのノードに配置することができません。

#### ノード仕様のテイントの例

```
spec:
  ....
  template:
    ....
    spec:
      taints:
        - effect: NoExecute
          key: key1
          value: value1
    ....
```

#### Pod 仕様での容認の例

```
spec:
  ....
  template:
    ....
    spec
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoExecute"
          tolerationSeconds: 3600
    ....
```

テイントおよび容認は、key、value、および effect で設定されています。

表2.1 テイントおよび容認コンポーネント

パラメーター	説明						
<b>key</b>	<b>key</b> には、253 文字までの文字列を使用できます。キーは文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。						
<b>value</b>	<b>value</b> には、63 文字までの文字列を使用できます。値は文字または数字で開始する必要があり、文字、数字、ハイフン、ドットおよびアンダースコアを含めることができます。						
<b>effect</b>	effect は以下のいずれかにすることができます。 <table border="1"> <tr> <td><b>NoSchedule</b> <sup>[1]</sup></td><td> <ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされません。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul> </td></tr> <tr> <td><b>PreferNoSchedule</b></td><td> <ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul> </td></tr> <tr> <td><b>NoExecute</b></td><td> <ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールできません。</li> <li>• 一致する容認を持たないノードの既存 Pod は削除されます。</li> </ul> </td></tr> </table>	<b>NoSchedule</b> <sup>[1]</sup>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされません。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul>	<b>PreferNoSchedule</b>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul>	<b>NoExecute</b>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールできません。</li> <li>• 一致する容認を持たないノードの既存 Pod は削除されます。</li> </ul>
<b>NoSchedule</b> <sup>[1]</sup>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされません。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul>						
<b>PreferNoSchedule</b>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールされる可能性があります、スケジューラーはスケジュールしないようにします。</li> <li>• ノードの既存 Pod はそのままになります。</li> </ul>						
<b>NoExecute</b>	<ul style="list-style-type: none"> <li>• テイントに一致しない新規 Pod はノードにスケジュールできません。</li> <li>• 一致する容認を持たないノードの既存 Pod は削除されます。</li> </ul>						
<b>operator</b>	<table border="1"> <tr> <td><b>Equal</b></td><td><b>key/value/effect</b> パラメーターは一致する必要があります。これはデフォルトになります。</td></tr> <tr> <td><b>Exists</b></td><td><b>key/effect</b> パラメーターは一致する必要があります。いずれかに一致する <b>value</b> パラメーターを空のままにする必要があります。</td></tr> </table>	<b>Equal</b>	<b>key/value/effect</b> パラメーターは一致する必要があります。これはデフォルトになります。	<b>Exists</b>	<b>key/effect</b> パラメーターは一致する必要があります。いずれかに一致する <b>value</b> パラメーターを空のままにする必要があります。		
<b>Equal</b>	<b>key/value/effect</b> パラメーターは一致する必要があります。これはデフォルトになります。						
<b>Exists</b>	<b>key/effect</b> パラメーターは一致する必要があります。いずれかに一致する <b>value</b> パラメーターを空のままにする必要があります。						

1. **NoSchedule** テイントをマスターノードに追加する場合、ノードには、デフォルトで追加される **node-role.kubernetes.io/master=:NoSchedule** テイントが必要です。  
以下は例になります。

```

apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0

```

```

machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  ...

```

容認はテイントと一致します。

- **operator** パラメーターが **Equal** に設定されている場合:
  - **key** パラメーターは同じになります。
  - **value** パラメーターは同じになります。
  - **effect** パラメーターは同じになります。
- **operator** パラメーターが **Exists** に設定されている場合:
  - **key** パラメーターは同じになります。
  - **effect** パラメーターは同じになります。

以下のテイントは OpenShift Container Platform に組み込まれています。

- **node.kubernetes.io/not-ready**: ノードは準備状態ではありません。これはノード条件 **Ready=False** に対応します。
- **node.kubernetes.io/unreachable**: ノードはノードコントローラーから到達不能です。これはノード条件 **Ready=Unknown** に対応します。
- **node.kubernetes.io/out-of-disk**: ノードには新しい Pod を追加するためのノード上の空きスペースが十分にありません。これはノード条件 **OutOfDisk=True** に対応します。
- **node.kubernetes.io/memory-pressure**: ノードにはメモリー不足の問題が発生しています。これはノード条件 **MemoryPressure=True** に対応します。
- **node.kubernetes.io/disk-pressure**: ノードにはディスク不足の問題が発生しています。これはノード条件 **DiskPressure=True** に対応します。
- **node.kubernetes.io/network-unavailable**: ノードのネットワークは使用できません。
- **node.kubernetes.io/unschedulable**: ノードはスケジュールが行えません。
- **node.cloudprovider.kubernetes.io/uninitialized**: ノードコントローラーが外部のクラウドプロバイダーを使って起動すると、このテイントはノード上に設定され、使用不可能とマークされます。cloud-controller-manager のコントローラーがこのノードを初期化した後に、kubelet がこのテイントを削除します。

#### 2.7.1.1. Pod のエビクションを遅延させる容認期間 (秒数) の使用方法

Pod 仕様または **MachineSet** に **tolerationSeconds** パラメーターを指定して、Pod がエビクションされる前にノードにバインドされる期間を指定できます。effect が **NoExecute** のテイントがノードに追加される場合、テイントを容認する Pod に **tolerationSeconds** パラメーターがある場合、Pod は期限切れになるまでエビクトされません。

## 出力例

```
spec:
....
  template:
....
    spec
      tolerations:
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoExecute"
        tolerationSeconds: 3600
```

ここで、この Pod が実行中であるものの、一致する容認がない場合、Pod は 3,600 秒間バインドされたままとなり、その後にエビクトされます。テイントが期限前に削除される場合、Pod はエビクトされません。

### 2.7.1.2. 複数のテイントの使用方法

複数のテイントを同じノードに、複数の容認を同じ Pod に配置することができます。OpenShift Container Platform は複数のテイントと容認を以下のように処理します。

- Pod に一致する容認のあるテイントを処理します。
- 残りの一致しないテイントは Pod について以下の effect を持ちます。
  - effect が **NoSchedule** の一致しないテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードにスケジュールできません。
  - effect が **NoSchedule** の一致しないテイントがなく、effect が **PreferNoSchedule** の一致しないテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod のノードへのスケジュールを試行しません。
  - effect が **NoExecute** のテイントが 1 つ以上ある場合、OpenShift Container Platform は Pod をノードからエビクトするか (ノードですでに実行中の場合)、または Pod のそのノードへのスケジュールが実行されません (ノードでまだ実行されていない場合)。
    - テイントを容認しない Pod はすぐにエビクトされます。
    - Pod の仕様に **tolerationSeconds** を指定せずにテイントを容認する Pod は永久にバインドされたままになります。
    - 指定された **tolerationSeconds** を持つテイントを容認する Pod は指定された期間バインドされます。

以下に例を示します。

- 以下のテイントをノードに追加します。

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- Pod には以下の容認があります。

```
spec:
....
  template:
....
    spec
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoSchedule"
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoExecute"
```

この場合、3つ目のテイントに一致する容認がないため、Pod はノードにスケジュールできません。Pod はこのテイントの追加時にノードですでに実行されている場合は実行が継続されます。3つ目のテイントは3つのテイントの中で Pod で容認されない唯一のテイントであるためです。

#### 2.7.1.3. Pod のスケジューリングとノードの状態 (Taint Nodes By Condition) について

Taint Nodes By Condition (状態別のノードへのテイント) 機能はデフォルトで有効にされており、これはメモリー不足やディスク不足などの状態を報告するノードを自動的にテイントします。ノードが状態を報告すると、その状態が解消するまでテイントが追加されます。テイントに **NoSchedule** の effect がある場合、ノードが一致する容認を持つまでそのノードに Pod をスケジュールすることはできません。

スケジューラーは、Pod をスケジュールする前に、ノードでこれらのテイントの有無をチェックします。テイントがある場合、Pod は別のノードにスケジュールされます。スケジューラーは実際のノードの状態ではなくテイントをチェックするので、適切な Pod 容認を追加して、スケジューラーがこのようなノードの状態を無視するように設定します。

デーモンセットコントローラーは、以下の容認をすべてのデーモンに自動的に追加し、下位互換性を確保します。

- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/out-of-disk (Critical Pod の場合のみ)
- node.kubernetes.io/unschedulable (1.10 以降)
- node.kubernetes.io/network-unavailable (ホストネットワークのみ)

デーモンセットには任意の容認を追加することも可能です。

#### 2.7.1.4. Pod の状態別エビクションについて (Taint-Based Eviction)

Taint-Based Eviction 機能はデフォルトで有効にされており、これは **not-ready** および **unreachable** などの特定の状態にあるノードから Pod をエビクトします。ノードがこうした状態のいずれかになると、OpenShift Container Platform はテイントをノードに自動的に追加して、Pod のエビクトおよび別のノードでの再スケジュールを開始します。

Taint Based Eviction には **NoExecute** の effect があり、そのテイントを容認しない Pod はすぐにエビクトされ、これを容認する Pod はエビクトされません (Pod が **tolerationSeconds** パラメーターを使用しない場合に限ります)。

**tolerationSeconds** パラメーターを使用すると、ノード状態が設定されたノードに Pod がどの程度の期間バインドされるかを指定することができます。**tolerationSeconds** の期間後もこの状態が続くと、テイントはノードに残り続け、一致する容認を持つ Pod はエビクトされます。**tolerationSeconds** の期間前にこの状態が解消される場合、一致する容認を持つ Pod は削除されません。

値なしで **tolerationSeconds** パラメーターを使用する場合、Pod は not ready(準備未完了) および unreachable(到達不能) のノードの状態が原因となりエビクトされることはありません。



### 注記

OpenShift Container Platform は、レートが制限された方法で Pod をエビクトし、マスターがノードからパーティション化される場合などのシナリオで発生する大規模な Pod エビクシオンを防ぎます。

OpenShift Container Platform は、**node.kubernetes.io/not-ready** および **node.kubernetes.io/unreachable** の容認を、**Pod** 設定がいずれかの容認を指定しない限り、自動的に **tolerationSeconds=300** に追加します。

```
spec:
  ....
  template:
    ....
    spec
      tolerations:
        - key: node.kubernetes.io/not-ready
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300 ①
        - key: node.kubernetes.io/unreachable
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300
```

- ① これらの容認は、ノード状態の問題のいずれかが検出された後、デフォルトの Pod 動作のバインドを 5 分間維持できるようにします。

これらの容認は必要に応じて設定できます。たとえば、アプリケーションに多数のローカル状態がある場合、ネットワークのパーティション化などに伴い、Pod をより長い時間ノードにバインドさせる必要があるかもしれません。これにより、パーティションを回復させることができ、Pod のエビクシオンを回避できます。

デーモンセットによって起動する Pod は、**tolerationSeconds** が指定されない以下のテイントの **NoExecute** 容認を使用して作成されます。

- **node.kubernetes.io/unreachable**
- **node.kubernetes.io/not-ready**

その結果、デーモンセット Pod は、これらのノードの状態が原因でエビクトされることはありません。



### 2.7.1.5. すべてのテイントの許容

ノードは、**operator: "Exists"** 容認を **key** および **value** パラメーターなしで追加することですべてのテイントを容認するように Pod を設定できます。この容認のある Pod はテイントを持つノードから削除されません。

#### すべてのテイントを容認するための Pod 仕様

```
spec:
....
  template:
....
    spec
      tolerations:
        - operator: "Exists"
```

### 2.7.2. テイントおよび容認 (Toleration) の追加

容認を Pod に、テイントをノードに追加することで、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) Pod を制御できます。既存の Pod およびノードの場合、最初に容認を Pod に追加してからテイントをノードに追加して、容認を追加する前に Pod がノードから削除されないようにする必要があります。

#### 手順

1. **Pod** 仕様を **tolerations** スタンザを含めるように編集して、容認を Pod に追加します。

#### Equal 演算子を含む Pod 設定ファイルのサンプル

```
spec:
....
  template:
....
    spec:
      tolerations:
        - key: "key1" ❶
          value: "value1"
          operator: "Equal"
          effect: "NoExecute"
          tolerationSeconds: 3600 ❷
```

- ❶ テイントおよび容認コンポーネント の表で説明されている toleration パラメーターです。
- ❷ **tolerationSeconds** パラメーターは、エビクトする前に Pod をどの程度の期間ノードにバインドさせるかを指定します。

以下に例を示します。

#### Exists 演算子を含む Pod 設定ファイルのサンプル

```
spec:
....
  template:
```

```
....
spec:
  tolerations:
    - key: "key1"
      operator: "Exists" ❶
      effect: "NoExecute"
      tolerationSeconds: 3600
```

- ❶ **Exists** Operator は **value** を取りません。

この例では、テイントを、キー **key1**、値 **value1**、およびテイント effect **NoExecute** を持つ **node1** にテイントを配置します。

2. **テイントおよび容認コンポーネント** の表で説明されているパラメーターと共に以下のコマンドを使用してテイントをノードに追加します。

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

以下に例を示します。

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

このコマンドは、キー **key1**、値 **value1**、および effect **NoExecute** を持つテイントを **node1** に配置します。

### 注記

**NoSchedule** テイントをマスターノードに追加する場合、ノードには、デフォルトで追加される **node-role.kubernetes.io/master=:NoSchedule** テイントが必要です。

以下は例になります。

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-
v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
...
```

Pod の容認はノードのテイントに一致します。いずれかの容認のある Pod は **node1** にスケジューリングできます。

### 2.7.3. マシンセットを使用したテイントおよび容認の追加

マシンセットを使用してテイントをノードに追加できます。**MachineSet** オブジェクトに関連付けられるすべてのノードがテイントで更新されます。容認は、ノードに直接追加されたテイントと同様に、マシンセットによって追加されるテイントに応答します。

## 手順

1. **Pod** 仕様を **tolerations** スタンザを含めるように編集して、容認を Pod に追加します。

### Equal 演算子を含む Pod 設定ファイルのサンプル

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: "key1" ❶
          value: "value1"
          operator: "Equal"
          effect: "NoExecute"
          tolerationSeconds: 3600 ❷
```

- ❶ テイントおよび容認コンポーネント の表で説明されている toleration パラメーターです。
- ❷ **tolerationSeconds** パラメーターは、エビクトする前に Pod をどの程度の期間ノードにバインドさせるかを指定します。

以下に例を示します。

### Exists 演算子を含む Pod 設定ファイルのサンプル

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: "key1"
          operator: "Exists"
          effect: "NoExecute"
          tolerationSeconds: 3600
```

2. テイントを **MachineSet** オブジェクトに追加します。
  - a. テイントを付けるノードの **MachineSet** YAML を編集するか、または新規 **MachineSet** オブジェクトを作成できます。

```
$ oc edit machineset <machineset>
```

- b. テイントを **spec.template.spec** セクションに追加します。

### ノード仕様のテイントの例

```
spec:
  ....
  template:
    ....
    spec:
      taints:
        - effect: NoExecute
          key: key1
          value: value1
      ....
```

この例では、キー **key1**、値 **value1**、およびテイント effect **NoExecute** を持つテイントをノードに配置します。

- c. 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
```

マシンが削除されるまで待機します。

- d. 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
```

マシンが起動するまで待ちます。テイントは **MachineSet** オブジェクトに関連付けられたノードに追加されます。

## 2.7.4. テイントおよび容認 (Toleration) 使ってユーザーをノードにバインドする

ノードのセットを特定のユーザーセットによる排他的な使用のために割り当てる必要がある場合、容認をそれらの Pod に追加します。次に、対応するテイントをそれらのノードに追加します。容認が設定された Pod は、テイントが付けられたノードまたはクラスター内の他のノードを使用できます。

Pod がテイントが付けられたノードのみにスケジュールされるようにするには、ラベルを同じノードセットに追加し、ノードのアフィニティーを Pod に追加し、Pod がそのラベルの付いたノードのみにスケジュールできるようにします。

### 手順

ノードをユーザーの使用可能な唯一のノードとして設定するには、以下を実行します。

1. 対応するテイントをそれらのノードに追加します。  
以下に例を示します。

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

2. カスタム受付コントローラーを作成して容認を Pod に追加します。

## 2.7.5. テイントおよび容認 (Toleration) を使って特殊ハードウェアを持つノードを制御する

ノードの小規模なサブセットが特殊ハードウェアを持つクラスターでは、テイントおよび容認 (Toleration) を使用して、特殊ハードウェアを必要としない Pod をそれらのノードから切り離し、特殊ハードウェアを必要とする Pod をそのままにすることができます。また、特殊ハードウェアを必要と

する Pod に対して特定のノードを使用することを要求することもできます。

これは、特殊ハードウェアを必要とする Pod に容認を追加し、特殊ハードウェアを持つノードにテイントを付けることで実行できます。

## 手順

特殊ハードウェアを持つノードが特定の Pod 用に予約されるようにするには、以下を実行します。

1. 容認を特別なハードウェアを必要とする Pod に追加します。  
以下に例を示します。

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: "disktype"
          value: "ssd"
          operator: "Equal"
          effect: "NoSchedule"
          tolerationSeconds: 3600
```

2. 以下のコマンドのいずれかを使用して、特殊ハードウェアを持つノードにテイントを設定します。

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

または、以下を実行します。

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

### 2.7.6. テイントおよび容認 (Toleration) の削除

必要に応じてノードからテイントを、Pod から容認をそれぞれ削除できます。最初に容認を Pod に追加してからテイントをノードに追加して、容認を追加する前に Pod がノードから削除されないようにする必要があります。

## 手順

テイントおよび容認 (Toleration) を削除するには、以下を実行します。

1. ノードからテイントを削除するには、以下を実行します。

```
$ oc adm taint nodes <node-name> <key>-
```

以下に例を示します。

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

## 出力例

```
node/ip-10-0-132-248.ec2.internal untainted
```

- Pod から容認を削除するには、容認を削除するための **Pod** 仕様を編集します。

```
spec:
....
  template:
....
    spec:
      tolerations:
        - key: "key2"
          operator: "Exists"
          effect: "NoExecute"
          tolerationSeconds: 3600
```

## 2.8. TOPOLOGY MANAGER

Topology Manager について理解し、これを使用します。

### 2.8.1. Topology Manager ポリシー

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



#### 注記

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

Topology Manager は、**cpumanager-enabled** カスタムリソース (CR) で割り当てる 4 つの割り当てポリシーをサポートします。

#### none ポリシー

これはデフォルトのポリシーで、トポロジーの配置は実行しません。

#### best-effort ポリシー

**best-effort** トポロジー管理ポリシーを持つ Pod のそれぞれのコンテナの場合、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、そのコンテナの推奨される NUMA ノードのアフィニティを保存します。アフィニティ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、そのコンテナの推奨される NUMA ノードのアフィニティを保存します。アフィニティ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。

## 2.8.2. 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

Topology 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 ❶
  ...
```

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

- ❷ **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 ❶
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node ❷
```

- ❶ このパラメーターは **static** である必要があります。
- ❷ 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。

### 2.8.3. Pod の Topology Manager ポリシーとの対話

以下のサンプル **Pod** 仕様は、Pod の Topology Manager 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йс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を検出します。

Topology Manager はこの情報を使用して、このコンテナに最適なトポロジを保管します。この Pod の場合、CPU マネージャーおよびデバйс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。

## 2.9. リソース要求とオーバーコミット

各コンピュータリソースについて、コンテナはリソース要求および制限を指定できます。スケジューリングの決定は要求に基づいて行われ、ノードに要求される値を満たす十分な容量があることが確認されます。コンテナが制限を指定するものの、要求を省略する場合、要求はデフォルトで制限値に設定されます。コンテナは、ノードの指定される制限を超えることはできません。

制限の実施方法は、コンピュータリソースのタイプによって異なります。コンテナが要求または制限を指定しない場合、コンテナはリソース保証のない状態でノードにスケジュールされます。実際に、コンテナはローカルの最も低い優先順位で利用できる指定リソースを消費できます。リソースが不足する状態では、リソース要求を指定しないコンテナに最低レベルの QoS (Quality of Service) が設定されます。

スケジューリングは要求されるリソースに基づいて行われる一方で、クォータおよびハード制限はリソース制限のことを指しており、これは要求されるリソースよりも高い値に設定できます。要求と制限の間の差異は、オーバーコミットのレベルを定めるものとなります。たとえば、コンテナに 1Gi のメモリー要求と 2Gi のメモリー制限が指定される場合、コンテナのスケジューリングはノードで 1Gi を利用可能とする要求に基づいて行われますが、2Gi まで使用することができます。そのため、この場合のオーバーコミットは 200% になります。

## 2.10. CLUSTER RESOURCE OVERRIDE OPERATOR を使用したクラスターレベルのオーバーコミット

Cluster Resource Override Operator は、クラスター内のすべてのノードでオーバーコミットのレベルを制御し、コンテナの密度を管理できる受付 Webhook です。Operator は、特定のプロジェクトのノードが定義されたメモリーおよび CPU 制限を超える場合について制御します。

以下のセクションで説明されているように、OpenShift Container Platform コンソールまたは CLI を使用して Cluster Resource Override Operator をインストールする必要があります。インストール時に、以下の例のように、オーバーコミットのレベルを設定する **ClusterResourceOverride** カスタムリソース (CR) を作成します。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
- name: cluster 1
spec:
  memoryRequestToLimitPercent: 50 2
  cpuRequestToLimitPercent: 25 3
  limitCPUToMemoryPercent: 200 4
```

- 1** 名前は **cluster** でなければなりません。
- 2** オプション:コンテナのメモリー制限が指定されているか、またはデフォルトに設定されている場合、メモリー要求は制限のパーセンテージ (1-100) に対して上書きされます。デフォルトは 50 です。
- 3** オプション:コンテナの CPU 制限が指定されているか、またはデフォルトに設定されている場合、CPU 要求は、1-100 までの制限のパーセンテージに対応して上書きされます。デフォルトは 25 です。
- 4** オプション:コンテナのメモリー制限が指定されているか、デフォルトに設定されている場合、CPU 制限は、指定されている場合にメモリーのパーセンテージに対して上書きされます。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。



### 注記

Cluster Resource Override Operator の上書きは、制限がコンテナに設定されていない場合は影響を与えません。個別プロジェクトごとのデフォルト制限を使用して **LimitRange** オブジェクトを作成するか、または **Pod** 仕様で制限を設定し、上書きが適用されるようにします。

設定時に、以下のラベルを各プロジェクトの namespace オブジェクトに適用し、上書きをプロジェクトごとに有効にできます。

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

labels:
  clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"

....
```

Operator は **ClusterResourceOverride** CR の有無を監視し、**ClusterResourceOverride** 受付 Webhook が Operator と同じ namespace にインストールされるようにします。

### 2.10.1. Web コンソールを使用した Cluster Resource Override Operator のインストール

クラスターでオーバーコミットを制御できるように、OpenShift Container Platform Web コンソールを使用して Cluster Resource Override Operator をインストールできます。

#### 前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。**LimitRange** オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、または **Pod** 仕様で制限を設定して上書きが適用されるようにする必要があります。

#### 手順

OpenShift Container Platform Web コンソールを使って Cluster Resource Override Operator をインストールするには、以下を実行します。

- OpenShift Container Platform Web コンソールで、**Home** → **Projects** に移動します。
  - Create Project** をクリックします。
  - clusterresourceoverride-operator** をプロジェクトの名前として指定します。
  - Create** をクリックします。
- Operators** → **OperatorHub** に移動します。
  - 利用可能な Operator の一覧から **ClusterResourceOverride Operator** を選択し、**Install** をクリックします。
  - Install Operator** ページで、**A specific Namespace on the cluster**が **Installation Mode** について選択されていることを確認します。
  - clusterresourceoverride-operator** が **Installed Namespace** について選択されていることを確認します。
  - Update Channel** および **Approval Strategy** を選択します。
  - Install** をクリックします。
- Installed Operators** ページで、**ClusterResourceOverride** をクリックします。
  - ClusterResourceOverride Operator** の詳細ページで、**Create Instance** をクリックします。
  - Create ClusterResourceOverride** ページで、YAML テンプレートを編集して、必要に応じてオーバーコミット値を設定します。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
```

```
memoryRequestToLimitPercent: 50 2
cpuRequestToLimitPercent: 25 3
limitCPUMemoryPercent: 200 4
```

- 1 名前は **cluster** でなければなりません。
  - 2 オプション:コンテナメモリの制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 50 です。
  - 3 オプション:コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。
  - 4 オプション:コンテナメモリの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。
- c. **Create** をクリックします。
4. クラスターカスタムリソースのステータスをチェックして、受付 Webhook の現在の状態を確認します。
- a. **ClusterResourceOverride Operator** ページで、**cluster** をクリックします。
  - b. **ClusterResourceOverride Details** ページで、**YAML** をクリックします。Webhook の呼び出し時に、**mutatingWebhookConfigurationRef** セクションが表示されます。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","met
adata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUMemoryPercent":200,"memoryRequestToLi
mitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

....

mutatingWebhookConfigurationRef: 1
  apiVersion: admissionregistration.k8s.io/v1beta1
```

```
kind: MutatingWebhookConfiguration
name: clusterresourceoverrides.admission.autoscaling.openshift.io
resourceVersion: "127621"
uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3
....
```

### 1 ClusterResourceOverride 受付 Webhook への参照。

## 2.10.2. CLI を使用した Cluster Resource Override Operator のインストール

OpenShift Container Platform CLI を使用して Cluster Resource Override Operator をインストールし、クラスターでのオーバーコミットを制御できます。

### 前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。**LimitRange** オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、または **Pod** 仕様で制限を設定して上書きが適用されるようにする必要があります。

### 手順

CLI を使用して Cluster Resource Override Operator をインストールするには、以下を実行します。

- Cluster Resource Override の namespace を作成します。
  - Cluster Resource Override Operator の **Namespace** オブジェクト YAML ファイル (**cro-namespace.yaml** など) を作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: clusterresourceoverride-operator
```

- namespace を作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

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

- Operator グループを作成します。
  - Cluster Resource Override Operator の **OperatorGroup** オブジェクトの YAML ファイル (**cro-og.yaml** など) を作成します。

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

```
spec:
  targetNamespaces:
    - clusterresourceoverride-operator
```

- b. Operator グループを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-og.yaml
```

3. サブスクリプションを作成します。

- a. Cluster Resource Override Operator の **Subscription** オブジェクト YAML ファイル (cro-sub.yaml など) を作成します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "4.5"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- b. サブスクリプションを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

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

4. **ClusterResourceOverride** カスタムリソース (CR) オブジェクトを **clusterresourceoverride-operator** namespace に作成します。

- a. **clusterresourceoverride-operator** namespace に切り替えます。

```
$ oc project clusterresourceoverride-operator
```

- b. Cluster Resource Override Operator の **ClusterResourceOverride** オブジェクト YAML ファイル (cro-cr.yaml など) を作成します。

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
```

```
memoryRequestToLimitPercent: 50 2
cpuRequestToLimitPercent: 25 3
limitCPUToMemoryPercent: 200 4
```

- 1 名前は **cluster** でなければなりません。
- 2 オプション:コンテナメモリーの制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 50 です。
- 3 オプション:コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。
- 4 オプション:コンテナメモリーの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。

c. **ClusterResourceOverride** オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f cro-cr.yaml
```

5. クラスターカスタムリソースのステータスをチェックして、受付 Webhook の現在の状態を確認します。

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

Webhook の呼び出し時に、**mutatingWebhookConfigurationRef** セクションが表示されます。

## 出力例

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
```



```

cpuRequestToLimitPercent: 25
limitCPUToMemoryPercent: 200
memoryRequestToLimitPercent: 50
status:
....

mutatingWebhookConfigurationRef: ❶
apiVersion: admissionregistration.k8s.io/v1beta1
kind: MutatingWebhookConfiguration
name: clusterresourceoverrides.admission.autoscaling.openshift.io
resourceVersion: "127621"
uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3
....

```

- ❶ **ClusterResourceOverride** 受付 Webhook への参照。

### 2.10.3. クラスターレベルのオーバーコミットの設定

Cluster Resource Override Operator には、Operator がオーバーコミットを制御する必要のある各プロジェクトの **ClusterResourceOverride** カスタムリソース (CR) およびラベルが必要です。

#### 前提条件

- 制限がコンテナに設定されていない場合、Cluster Resource Override Operator は影響を与えません。 **LimitRange** オブジェクトを使用してプロジェクトのデフォルト制限を指定するか、または **Pod** 仕様で制限を設定して上書きが適用されるようにする必要があります。

#### 手順

クラスターレベルのオーバーコミットを変更するには、以下を実行します。

1. **ClusterResourceOverride** CR を編集します。

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
- name: cluster
spec:
  memoryRequestToLimitPercent: 50 ❶
  cpuRequestToLimitPercent: 25 ❷
  limitCPUToMemoryPercent: 200 ❸

```

- ❶ オプション:コンテナメモリーの制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 50 です。
- ❷ オプション:コンテナ CPU の制限を上書きするためのパーセンテージが使用される場合は、これを 1-100 までの値で指定します。デフォルトは 25 です。
- ❸ オプション:コンテナメモリーの制限を上書きするためのパーセンテージが使用される場合は、これを指定します。1Gi の RAM の 100 パーセントでのスケーリングは、1 CPU コアに等しくなります。これは、CPU 要求を上書きする前に処理されます (設定されている場合)。デフォルトは 200 です。



- 以下のラベルが Cluster Resource Override Operator がオーバーコミットを制御する必要のある各プロジェクトの namespace オブジェクトに追加されていることを確認します。

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

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" ❶
  ....
```

- ❶ このラベルを各プロジェクトに追加します。

## 2.11. ノードレベルのオーバーコミット

QoS (Quality of Service) 保証、CPU 制限、またはリソースの予約など、特定ノードでオーバーコミットを制御するさまざまな方法を使用できます。特定のノードおよび特定のプロジェクトのオーバーコミットを無効にすることもできます。

### 2.11.1. コンピュートリソースとコンテナについて

コンピュートリソースについてのノードで実施される動作は、リソースタイプによって異なります。

#### 2.11.1.1. コンテナの CPU 要求について

コンテナには要求する CPU の量が保証され、さらにコンテナで指定される任意の制限までノードで利用可能な CPU を消費できます。複数のコンテナが追加の CPU の使用を試行する場合、CPU 時間が各コンテナで要求される CPU の量に基づいて分配されます。

たとえば、あるコンテナが 500m の CPU 時間を要求し、別のコンテナが 250m の CPU 時間を要求した場合、ノードで利用可能な追加の CPU 時間は 2:1 の比率でコンテナ間で分配されます。コンテナが制限を指定している場合、指定した制限を超えて CPU を使用しないようにスロットリングされます。CPU 要求は、Linux カーネルの CFS 共有サポートを使用して適用されます。デフォルトで、CPU 制限は、Linux カーネルの CFS クォータサポートを使用して 100ms の測定間隔で適用されます。ただし、これは無効にすることができます。

#### 2.11.1.2. コンテナのメモリー要求について

コンテナには要求するメモリー量が保証されます。コンテナは要求したよりも多くのメモリーを使用できますが、いったん要求した量を超えた場合には、ノードのメモリーが不足している状態では強制終了される可能性があります。コンテナが要求した量よりも少ないメモリーを使用する場合、システムタスクやデーモンがノードのリソース予約で確保されている分よりも多くのメモリーを必要としない限りそれが強制終了されることはありません。コンテナがメモリーの制限を指定する場合、その制限量を超えると即時に強制終了されます。

### 2.11.2. オーバーコミットメントと QoS (Quality of Service) クラスについて

ノードは、要求を指定しない Pod がスケジュールされている場合やノードのすべての Pod での制限の合計が利用可能なマシンの容量を超える場合に **オーバーコミット** されます。

オーバーコミットされる環境では、ノード上の Pod がいずれかの時点で利用可能なコンピュートリソースよりも多くの量の使用を試行することができます。これが生じると、ノードはそれぞれの Pod に優先順位を指定する必要があります。この決定を行うために使用される機能は、QoS (Quality of Service) クラスと呼ばれます。

各コンピュートリソースについて、コンテナは 3 つの QoS クラスに分類されます (優先順位は降順)。

表2.2 QoS (Quality of Service) クラス

優先順位	クラス名	説明
1(最高)	<b>Guaranteed</b>	制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しい場合、コンテナは <b>Guaranteed</b> として分類されます。
2	<b>Burstable</b>	制限およびオプションの要求がすべてのリソースについて設定されている場合 (0 と等しくない) でそれらの値が等しくない場合、コンテナは <b>Burstable</b> として分類されます。
3(最低)	<b>BestEffort</b>	要求および制限がリソースのいずれについても設定されない場合、コンテナは <b>BestEffort</b> として分類されます。

メモリーは圧縮できないリソースであるため、メモリー不足の状態では、最も優先順位の低いコンテナが最初に強制終了されます。

- **Guaranteed** コンテナは優先順位が最も高いコンテナとして見なされ、保証されます。強制終了されるのは、これらのコンテナで制限を超えるか、またはシステムがメモリー不足の状態にあるものの、エビクトできる優先順位の低いコンテナが他にない場合のみです。
- システム不足の状態にある **Burstable** コンテナは、制限を超過し、**BestEffort** コンテナが他に存在しない場合に強制終了される可能性があります。
- **BestEffort** コンテナは優先順位の最も低いコンテナとして処理されます。これらのコンテナのプロセスは、システムがメモリー不足になると最初に強制終了されます。

#### 2.11.2.1. Quality of Service (QoS) 層でのメモリーの予約方法について

**qos-reserved** パラメーターを使用して、特定の QoS レベルの Pod で予約されるメモリーのパーセンテージを指定することができます。この機能は、最も低い QoS クラスの Pod が高い QoS クラスの Pod で要求されるリソースを使用できないようにするために要求されたリソースの予約を試行します。

OpenShift Container Platform は、以下のように **qos-reserved** パラメーターを使用します。

- **qos-reserved=memory=100%** の値は、**Burstable** および **BestEffort** QoS クラスが、これらより高い QoS クラスで要求されたメモリーを消費するのを防ぎます。これにより、**Guaranteed** および **Burstable** ワークロードのメモリーリソースの保証レベルを上げることが優先され、**BestEffort** および **Burstable** ワークロードでの OOM が発生するリスクが高まります。
- **qos-reserved=memory=50%** の値は、**Burstable** および **BestEffort** QoS クラスがこれらより高い QoS クラスによって要求されるメモリーの半分を消費することを許可します。
- **qos-reserved=memory=0%** の値は、**Burstable** および **BestEffort** QoS クラスがノードの割り

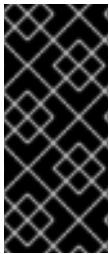
当て可能分を完全に消費することを許可しますが (利用可能な場合)、これにより、**Guaranteed** ワークロードが要求したメモリーにアクセスできなくなるリスクが高まります。この状況により、この機能は無効にされています。

### 2.11.3. swap メモリーと QOS について

QoS (Quality of Service) 保証を維持するため、swap はノード上でデフォルトで無効にすることができます。そうしない場合、ノードの物理リソースがオーバーサブスクライブし、Pod の配置時の Kubernetes スケジューラーによるリソース保証が影響を受ける可能性があります。

たとえば、2 つの Guaranteed pod がメモリー制限に達した場合、それぞれのコンテナが swap メモリーを使用し始める可能性があります。十分な swap 領域がない場合には、pod のプロセスはシステムのオーバーサブスクライブのために終了する可能性があります。

swap を無効にしないと、ノードが **MemoryPressure** にあることを認識しなくなり、Pod がスケジューリング要求に対応するメモリーを受け取れなくなります。結果として、追加の Pod がノードに配置され、メモリー不足の状態が加速し、最終的にはシステムの Out Of Memory (OOM) イベントが発生するリスクが高まります。



#### 重要

swap が有効にされている場合、利用可能なメモリーについてのリソース不足の処理 (out of resource handling) のエビクションしきい値は予期どおりに機能しなくなります。メモリー不足の状態の場合に Pod をノードからエビクトし、Pod を不足状態にない別のノードで再スケジューリングできるようにリソース不足の処理 (out of resource handling) を利用できるようにします。

### 2.11.4. ノードのオーバーコミットについて

オーバーコミット環境では、最適なシステム動作を提供できるようにノードを適切に設定する必要があります。

ノードが起動すると、メモリー管理用のカーネルの調整可能なフラグが適切に設定されます。カーネルは、物理メモリーが不足しない限り、メモリーの割り当てに失敗することはありません。

この動作を確認するため、OpenShift Container Platform は、**vm.overcommit\_memory** パラメーターを **1** に設定し、デフォルトのオペレーティングシステムの設定を上書きすることで、常にメモリーをオーバーコミットするようにカーネルを設定します。

また、OpenShift Container Platform は **vm.panic\_on\_oom** パラメーターを **0** に設定することで、メモリーが不足したときでもカーネルがパニックにならないようにします。0 の設定は、Out of Memory (OOM) 状態のときに oom\_killer を呼び出すようカーネルに指示します。これにより、優先順位に基づいてプロセスを強制終了します。

現在の設定は、ノードに以下のコマンドを実行して表示できます。

```
$ sysctl -a |grep commit
```

#### 出力例

```
vm.overcommit_memory = 1
```

```
$ sysctl -a |grep panic
```

## 出力例

```
vm.panic_on_oom = 0
```



### 注記

上記のフラグはノード上にすでに設定されているはずであるため、追加のアクションは不要です。

各ノードに対して以下の設定を実行することもできます。

- CPU CFS クォータを使用した CPU 制限の無効化または実行
- システムプロセスのリソース予約
- Quality of Service (QoS) 層でのメモリー予約

### 2.11.5. CPU CFS クォータの使用による CPU 制限の無効化または実行

デフォルトで、ノードは Linux カーネルの Completely Fair Scheduler (CFS) クォータのサポートを使用して、指定された CPU 制限を実行します。

CPU 制限の適用を無効にする場合、それがノードに与える影響を理解しておくことが重要になります。

- コンテナに CPU 要求がある場合、これは Linux カーネルの CFS 共有によって引き続き適用されます。
- コンテナに CPU 要求がなく、CPU 制限がある場合は、CPU 要求はデフォルトで指定される CPU 制限に設定され、Linux カーネルの CFS 共有によって適用されます。
- コンテナに CPU 要求と制限の両方がある場合、CPU 要求は Linux カーネルの CFS 共有によって適用され、CPU 制限はノードに影響を与えません。

### 前提条件

1. 設定するノードタイプの静的な **MachineConfigPool** CRD に関連付けられたラベルを取得します。以下のいずれかの手順を実行します。
  - a. マシン設定プールを表示します。

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

以下に例を示します。

```
$ oc describe machineconfigpool worker
```

### 出力例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
```

```
generation: 1
labels:
  custom-kubelet: small-pods ❶
```

- ❶ ラベルが追加されると、**labels** の下に表示されます。

- b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

## 手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

### CPU 制限を無効化する設定例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods ❷
  kubeletConfig:
    cpuCfsQuota: ❸
      - "false"
```

- ❶ CR に名前を割り当てます。
- ❷ 設定の変更を適用するラベルを指定します。
- ❸ **cpuCfsQuota** パラメーターを **false** に設定します。

## 2.11.6. システムリソースのリソース予約

より信頼できるスケジューリングを実現し、ノードリソースのオーバーコミットメントを最小化するために、各ノードでは、クラスターが機能できるようノードで実行する必要のあるシステムデーモン用にそのリソースの一部を予約することができます。とくに、メモリーなどの圧縮できないリソースのリソースを予約することが推奨されます。

## 手順

Pod 以外のプロセスのリソースを明示的に予約するには、スケジューリングで利用可能なリソースを指定することにより、ノードリソースを割り当てます。詳細については、ノードのリソースの割り当てを参照してください。

## 2.11.7. ノードのオーバーコミットの無効化

有効にされているオーバーコミットを、各ノードで無効にできます。

## 手順

ノード内のオーバーコミットを無効にするには、そのノード上で以下のコマンドを実行します。

```
$ sysctl -w vm.overcommit_memory=0
```

## 2.12. プロジェクトレベルの制限

オーバーコミットを制御するには、プロジェクトごとのリソース制限の範囲を設定し、オーバーコミットが超過できないプロジェクトのメモリーおよび CPU 制限およびデフォルト値を指定できます。

プロジェクトレベルのリソース制限の詳細は、関連情報を参照してください。

または、特定のプロジェクトのオーバーコミットを無効にすることもできます。

### 2.12.1. プロジェクトでのオーバーコミットメントの無効化

有効にされているオーバーコミットメントをプロジェクトごとに無効にすることができます。たとえば、インフラストラクチャーコンポーネントはオーバーコミットメントから独立して設定できます。

#### 手順

プロジェクト内のオーバーコミットメントを無効にするには、以下の手順を実行します。

1. プロジェクトのオブジェクトファイルを編集します。
2. 以下のアノテーションを追加します。

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

3. プロジェクトのオブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

## 2.13. ガベージコレクションを使用しているノードリソースの解放

ガベージコレクションについて理解し、これを使用します。

### 2.13.1. 終了したコンテナがガベージコレクションによって削除される仕組みについて

コンテナのガベージコレクションは、エビクションしきい値を使用して実行することができます。

エビクションしきい値がガベージコレクションに設定されていると、ノードは Pod のコンテナが API から常にアクセス可能な状態になるよう試みます。Pod が削除された場合、コンテナも削除されます。コンテナは Pod が削除されず、エビクションしきい値に達していない限り保持されます。ノードがディスク不足 (disk pressure) の状態になっていると、コンテナが削除され、それらのログは **oc logs** を使用してアクセスできなくなります。

- **eviction-soft** - ソフトエビクションのしきい値は、エビクションしきい値と要求される管理者指定の猶予期間を組み合わせます。
- **eviction-hard** - ハードエビクションのしきい値には猶予期間がなく、検知されると、OpenShift Container Platform はすぐにアクションを実行します。

ノードがソフトエビクションしきい値の上限と下限の間で変動し、その関連する猶予期間を超えていない場合、対応するノードは、**true** と **false** の間で常に変動します。したがって、スケジューラーは適切なスケジュールを決定できない可能性があります。

この変動から保護するには、**eviction-pressure-transition-period** フラグを使用して、OpenShift Container Platform が不足状態から移行するまでにかかる時間を制御します。OpenShift Container Platform は、false 状態に切り替わる前の指定された期間に、エビクションしきい値を指定された不足状態に一致するように設定しません。

### 2.13.2. ガベージコレクションによってイメージが削除される仕組みについて

イメージのガベージコレクションでは、ノードの **cAdvisor** によって報告されるディスク使用量に基づいて、ノードから削除するイメージを決定します。

イメージのガベージコレクションのポリシーは、以下の 2 つの条件に基づいています。

- イメージのガベージコレクションをトリガーするディスク使用量のパーセント (整数で表される) です。デフォルトは 85 です。
- イメージのガベージコレクションが解放しようとするディスク使用量のパーセント (整数で表される) です。デフォルトは 80 です。

イメージのガベージコレクションのために、カスタムリソースを使用して、次の変数のいずれかを変更することができます。

表2.3 イメージのガベージコレクションを設定するための変数

設定	説明
<b>imageMinimumGCAge</b>	ガベージコレクションによって削除されるまでの未使用のイメージの有効期間。デフォルトは、2m です。
<b>imageGCHighThresholdPercent</b>	イメージのガベージコレクションをトリガーするディスク使用量のパーセント (整数で表される) です。デフォルトは 85 です。
<b>imageGCLowThresholdPercent</b>	イメージのガベージコレクションが解放しようとするディスク使用量のパーセント (整数で表される) です。デフォルトは 80 です。

以下の 2 つのイメージ一覧がそれぞれのガベージコレクターの実行で取得されます。

1. 1 つ以上の Pod で現在実行されているイメージの一覧
2. ホストで利用可能なイメージの一覧

新規コンテナの実行時に新規のイメージが表示されます。すべてのイメージにはタイムスタンプのマークが付けられます。イメージが実行中 (上記の最初の一覧) か、または新規に検出されている (上記の 2 番目の一覧) 場合、これには現在の時間のマークが付けられます。残りのイメージには以前のタイムスタンプのマークがすでに付けられています。すべてのイメージはタイムスタンプで並び替えられます。

コレクションが開始されると、停止条件を満たすまでイメージが最も古いものから順番に削除されます。

### 2.13.3. コンテナおよびイメージのガベージコレクションの設定

管理者は、**kubeletConfig** オブジェクトを各マシン設定プール用に作成し、OpenShift Container Platform によるガベージコレクションの実行方法を設定できます。



### 注記

OpenShift Container Platform は、各マシン設定プールの **kubeletConfig** オブジェクトを1つのみサポートします。

次のいずれかの組み合わせを設定できます。

- コンテナのソフトエビクション
- コンテナのハードエビクション
- イメージのエビクション

ソフトコンテナエビクションの場合は、エビクションされるまでの猶予期間を設定することもできます。

### 前提条件

1. 設定するノードタイプの静的な **MachineConfigPool** CRD に関連付けられたラベルを取得します。以下のいずれかの手順を実行します。
  - a. マシン設定プールを表示します。

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

以下に例を示します。

```
$ oc describe machineconfigpool worker
```

### 出力例

```
Name:      worker
Namespace:
Labels:    custom-kubelet=small-pods 1
```

- 1 ラベルが追加されると、**Labels** の下に表示されます。

- b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

### 手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

#### コンテナのガベージコレクション CR のサンプル設定:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
```



```

metadata:
  name: worker-kubeconfig ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods ❷
  kubeletConfig:
    evictionSoft: ❸
      memory.available: "500Mi" ❹
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: ❺
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard:
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s ❻
    imageMinimumGCAge: 5m ❼
    imageGCHighThresholdPercent: 80 ❽
    imageGCLowThresholdPercent: 75 ❾

```

- ❶ オブジェクトの名前。
- ❷ セレクターラベル。
- ❸ エビクションのタイプ: **EvictionSoft** および **EvictionHard**。
- ❹ 特定のエビクショントリガーシグナルに基づくエビクションのしきい値。
- ❺ ソフトエビクションの猶予期間。このパラメーターは、**eviction-hard** には適用されません。
- ❻ エビクションの不足状態から移行するまでの待機時間
- ❼ ガベージコレクションによって削除されるまでの未使用のイメージの有効期間。
- ❽ イメージのガベージコレクションをトリガーするディスク使用量のパーセント (整数で表される) です。
- ❾ イメージのガベージコレクションが解放しようとするディスク使用量のパーセント (整数で表される) です。

## 2. オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml
```

以下に例を示します。

```
$ oc create -f gc-container.yaml
```

#### 出力例

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

3. ガベージコレクションがアクティブであることを確認します。カスタムリソースで指定した Machine Config Pool では、変更が完全に実行されるまで **UPDATING** が 'true' と表示されません。

```
$ oc get machineconfigpool
```

#### 出力例

NAME	CONFIG	UPDATED	UPDATING
master	rendered-master-546383f80705bd5aeaba93	True	False
worker	rendered-worker-b4c51bb33ccaae6fc4a6a5	False	True

## 2.14. NODE TUNING OPERATOR の使用

Node Tuning Operator について理解し、これを使用します。

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

Operator は、コンテナ化された 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。

### 2.14.1. 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せられます。



### 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。

## 2.14.2. 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: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
```

- ❶ オプション:
- ❷ 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。
- ❸ 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。
- ❹ オプションの一覧。
- ❺ 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)。
- ❻ 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> ❶
  value: <label_value> ❷
  type: <label_type> ❸
  <match> ❹
```

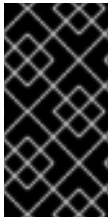
- ❶ ノードまたは Pod のラベル名。
- ❷ オプションのノードまたは Pod のラベルの値。省略されている場合も、**<label\_name>** があるだけで一致条件を満たします。
- ❸ オプションのオブジェクトタイプ (**node** または **pod**)。省略されている場合は、**node** が想定されます。
- ❹ オプションの **<match>** 一覧。

**<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** に評価される必要があります。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

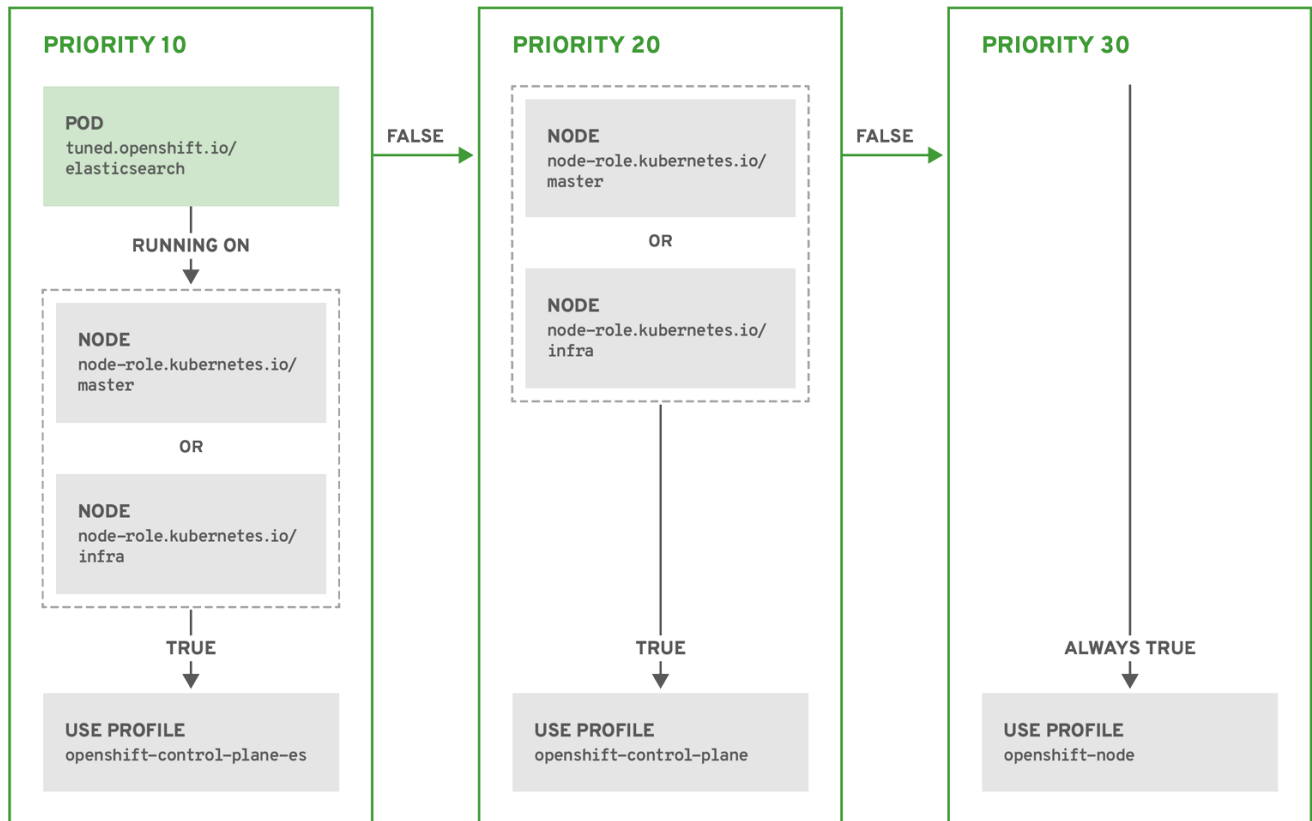
```
- match:
- label: tuned.openshift.io/elasticsearch
  match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  type: pod
priority: 10
profile: openshift-control-plane-es
- match:
- label: node-role.kubernetes.io/master
- label: node-role.kubernetes.io/infra
priority: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



OPENSIFT\_10\_0319

### 例: マシン設定プールベースのマッチング

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=Custom OpenShift node profile with an additional kernel parameter
        include=openshift-node
        [bootloader]
        cmdline_openshift_node_custom=+skew_tick=1
        name: openshift-node-custom
  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: "worker-custom"
    priority: 20
    profile: openshift-node-custom
```

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

### 2.14.3. 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

```

#### 2.14.4. 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。

## 2.15. ノードあたりの POD の最大数の設定

**podsPerCore**  および  **maxPods**  の 2 つのパラメーターはノードに対してスケジュールできる Pod の最大数を制御します。両方のオプションを使用した場合、より低い値の方がノード上の Pod の数を制限します。

たとえば、 **podsPerCore**  が 4 つのプロセッサコアを持つノード上で、 **10**  に設定されていると、ノード上で許容される Pod の最大数は 40 になります。

### 前提条件

1. 設定するノードタイプの静的な  **MachineConfigPool**  CRD に関連付けられたラベルを取得します。以下のいずれかの手順を実行します。
  - a. マシン設定プールを表示します。

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

以下に例を示します。

```
$ oc describe machineconfigpool worker
```

### 出力例

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

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

- b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

## 手順

1. 設定変更のためのカスタムリソース (CR) を作成します。

### max-pods CR の設定例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods ❷
  kubeletConfig:
    podsPerCore: 10 ❸
    maxPods: 250 ❹
```

- ❶ CR に名前を割り当てます。
- ❷ 設定の変更を適用するラベルを指定します。
- ❸ ノードがプロセッサコアの数に基づいて実行できる Pod の数を指定します。
- ❹ ノードのプロパティにかかわらず、ノードが実行できる Pod 数を固定値に指定します。



### 注記

**podsPerCore** を **0** に設定すると、この制限が無効になります。

上記の例では、**podsPerCore** のデフォルト値は **10** であり、**maxPods** のデフォルト値は **250** です。つまり、ノードのコア数が 25 以上でない限り、デフォルトにより **podsPerCore** が制限要素になります。

2. 変更が適用されるかどうかを確認するために、**MachineConfigPool** CRD を一覧表示します。変更が Machine Config Controller によって取得されると、**UPDATING** 列で **True** と報告されます。

```
$ oc get machineconfigpools
```

### 出力例

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	False	False
worker	worker-8cecd1236b33ee3f8a5e	False	True	False

変更が完了すると、**UPDATED** 列で **True** と報告されます。

```
$ oc get machineconfigpools
```

出力例

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	True	False
worker	worker-8cecd1236b33ee3f8a5e	True	False	False

## 第3章 インストール後のネットワーク設定

OpenShift Container Platform のインストール後に、ネットワークをさらに拡張し、要件に合わせてカスタマイズできます。

### 3.1. OPENSIFT SDN を使用したネットワークポリシーの設定

ネットワークポリシーについて理解し、これを使用します。

#### 3.1.1. ネットワークポリシーについて

Kubernetes ネットワークポリシーをサポートする Kubernetes Container Network Interface (CNI) プラグインを使用するクラスターでは、ネットワークの分離は **NetworkPolicy** オブジェクトによって完全に制御されます。OpenShift Container Platform 4.5 では、OpenShift SDN はデフォルトのネットワーク分離モードでのネットワークポリシーの使用をサポートしています。



#### 注記

OpenShift SDN クラスターネットワークプロバイダーを使用する場合、ネットワークポリシーについて、以下の制限が適用されます。

- **egress** フィールドで指定される egress ネットワークポリシーはサポートされていません。
- IPBlock はネットワークポリシーでサポートされますが、**except** 句はサポートしません。**except** 句を含む IPBlock セクションのあるポリシーを作成する場合、SDN Pod は警告をログに記録し、そのポリシーの IPBlock セクション全体は無視されます。



#### 警告

ネットワークポリシーは、ホストのネットワーク namespace には適用されません。ホストネットワークが有効にされている Pod はネットワークポリシールールによる影響を受けません。

デフォルトで、プロジェクトのすべての Pod は他の Pod およびネットワークのエンドポイントからアクセスできます。プロジェクトで1つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

Pod が1つ以上の **NetworkPolicy** オブジェクトのセレクターで一致する場合、Pod はそれらの1つ以上の **NetworkPolicy** オブジェクトで許可される接続のみを受け入れます。**NetworkPolicy** オブジェクトによって選択されていない Pod は完全にアクセス可能です。

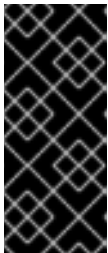
以下のサンプル **NetworkPolicy** オブジェクトは、複数の異なるシナリオをサポートすることを示しています。

- すべてのトラフィックを拒否します。

プロジェクトに deny by default (デフォルトで拒否) を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

- OpenShift Container Platform Ingress コントローラーからの接続のみを許可します。  
プロジェクトで OpenShift Container Platform Ingress コントローラーからの接続のみを許可するには、以下の **NetworkPolicy** オブジェクトを追加します。



### 重要

OVN-Kubernetes ネットワークプロバイダープラグインの場合、Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するように設定されている場合、Ingress トラフィックが許可され、他のすべてのトラフィックが拒否されるようにネットワークポリシーを適用するための方法はサポートされていません。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
                network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
    - Ingress
```

Ingress コントローラーが **endpointPublishingStrategy: HostNetwork** で設定されている場合、Ingress コントローラー Pod はホストネットワーク上で実行されます。ホストネットワーク上で実行されている場合、Ingress コントローラーからのトラフィックに **netid:0** Virtual Network ID (VNID) が割り当てられます。Ingress Operator に関連付けられる namespace の **netid** は異なるため、**allow-from-openshift-ingress** ネットワークポリシーの **matchLabel** は **default** Ingress コントローラーからのトラフィックに一致しません。OpenShift SDN では、**default** namespace に **netid:0** VNID が割り当てられるため、**default** namespace に **network.openshift.io/policy-group: ingress** でラベルを付けて、**default** Ingress コントローラーからのトラフィックを許可できます。

- プロジェクト内の Pod からの接続のみを受け入れます。  
Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下の **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
```

```

metadata:
  name: allow-same-namespace
spec:
  podSelector:
    ingress:
      - from:
        - podSelector: {}

```

- Pod ラベルに基づいて HTTP および HTTPS トラフィックのみを許可します。  
特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - ports:
      - protocol: TCP
        port: 80
      - protocol: TCP
        port: 443

```

- namespace および Pod セレクターの両方を使用して接続を受け入れます。  
namespace と Pod セレクターを組み合わせるネットワークトラフィックのマッチングをするには、以下と同様の **NetworkPolicy** オブジェクトを使用できます。

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
        podSelector:
          matchLabels:
            name: test-pods

```

**NetworkPolicy** オブジェクトは加算されるものです。つまり、複数の **NetworkPolicy** オブジェクトを組み合わせることで複雑なネットワーク要件を満たすことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、同じプロジェクト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend** の付いた Pod は各ポリシーで許可されるすべての接続を受け入れます。つま

り、同じ namespace の Pod からのすべてのポート、およびすべての namespace の Pod からのポート **80** および **443** での接続を受け入れます。

### 3.1.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
  matchLabels:
    app: mongodb
  ingress:
  - from:
    - podSelector: 3
      matchLabels:
        app: app
  ports: 4
  - protocol: TCP
    port: 27017
```

- 1** NetworkPolicy オブジェクトの **name**。
- 2** ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
- 3** ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。
- 4** トラフィックを受け入れる1つ以上の宛先の一覧。

### 3.1.3. ネットワークポリシーの作成

クラスターの namespace に許可される Ingress または egress ネットワークトラフィックを記述する詳細なルールを定義するには、ネットワークポリシーを作成できます。



#### 注記

**cluster-admin** ロールを持つユーザーでログインしている場合、クラスター内の namespace でネットワークポリシーを作成できます。

#### 前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

- ネットワークポリシーが適用される namespace で作業している。

## 手順

1. ポリシールールを作成します。

- a. **<policy\_name>.yaml** ファイルを作成します。

```
$ touch <policy_name>.yaml
```

ここでは、以下のようになります。

### **<policy\_name>**

ネットワークポリシーファイル名を指定します。

- b. 作成したばかりのファイルで、以下の例のようなネットワークポリシーを定義します。

**すべての namespace のすべての Pod から ingress を拒否します。**

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

**同じ namespace のすべての Pod から ingress を許可します。**

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
    - from:
      - podSelector: {}
```

2. ネットワークポリシーオブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

ここでは、以下のようになります。

### **<policy\_name>**

ネットワークポリシーファイル名を指定します。

### **<namespace>**

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

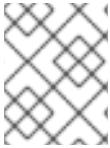
## 出力例



```
networkpolicy "default-deny" created
```

### 3.1.4. ネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。



#### 注記

**cluster-admin** ロールを持つユーザーでログインしている場合、クラスター内のネットワークポリシーを削除できます。

#### 前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

#### 手順

- **NetworkPolicy** オブジェクトを削除するには、以下のコマンドを入力します。

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

ここでは、以下ようになります。

#### <policy\_name>

ネットワークポリシーの名前を指定します。

#### <namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

#### 出力例

```
networkpolicy.networking.k8s.io/allow-same-namespace deleted
```

### 3.1.5. ネットワークポリシーの表示

namespace のネットワークポリシーを検査できます。



#### 注記

**cluster-admin** ロールを持つユーザーでログインしている場合、クラスター内のネットワークポリシーを表示できます。

#### 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

## 手順

- namespace のネットワークポリシーを一覧表示します。
  - namespace で定義された **NetworkPolicy** オブジェクトを表示するには、以下のコマンドを実行します。

```
$ oc get networkpolicy
```

- オプション: 特定のネットワークポリシーを検査するには、以下のコマンドを入力します。

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

ここでは、以下のようになります。

### <policy\_name>

検査するネットワークポリシーの名前を指定します。

### <namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

以下に例を示します。

```
$ oc describe networkpolicy allow-same-namespace
```

## oc describe コマンドの出力

```
Name:      allow-same-namespace
Namespace: ns1
Created on: 2021-05-24 22:28:56 -0400 EDT
Labels:    <none>
Annotations: <none>
Spec:
  PodSelector:  <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: <none>
  Not affecting egress traffic
  Policy Types: Ingress
```

## 3.1.6. ネットワークポリシーを使用したマルチテナント分離の設定

他のプロジェクト namespace の Pod およびサービスから分離できるようにプロジェクトを設定できます。

## 前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

## 手順

1. 以下の **NetworkPolicy** オブジェクトを作成します。
  - a. **allow-from-openshift-ingress** という名前のポリシー:



### 重要

OVN-Kubernetes ネットワークプロバイダープラグインの場合、Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するように設定されている場合、Ingress トラフィックが許可され、他のすべてのトラフィックが拒否されるようにネットワークポリシーを適用するための方法はサポートされていません。

```
$ cat << EOF | oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
    podSelector: {}
  policyTypes:
  - Ingress
EOF
```

- b. **allow-from-openshift-monitoring** という名前のポリシー。

```
$ cat << EOF | oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
    podSelector: {}
```

```
policyTypes:
- Ingress
EOF
```

- c. **allow-same-namespace** という名前のポリシー:

```
$ cat << EOF | oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

2. **default** Ingress コントローラー設定に **spec.endpointPublishingStrategy: HostNetwork** の値が設定されている場合、ラベルを **default** OpenShift Container Platform namespace に適用し、Ingress コントローラーとプロジェクト間のネットワークトラフィックを許可する必要があります。

- a. **default** Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するかどうかを判別します。

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
--output jsonpath='{.status.endpointPublishingStrategy.type}'
```

- b. 直前のコマンドによりエンドポイント公開ストラテジーが **HostNetwork** として報告される場合には、**default** namespace にラベルを設定します。

```
$ oc label namespace default 'network.openshift.io/policy-group=ingress'
```

3. 以下のコマンドを実行し、**NetworkPolicy** オブジェクトが現在のプロジェクトに存在することを確認します。

```
$ oc get networkpolicy <policy-name> -o yaml
```

以下の例では、**allow-from-openshift-ingress NetworkPolicy** オブジェクトが表示されています。

```
$ oc get -n project1 networkpolicy allow-from-openshift-ingress -o yaml
```

## 出力例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
  namespace: project1
spec:
  ingress:
  - from:
```

```
- namespaceSelector:
  matchLabels:
    network.openshift.io/policy-group: ingress
podSelector: {}
policyTypes:
- Ingress
```

### 3.1.7. 新規プロジェクトのデフォルトネットワークポリシーの作成

クラスター管理者は、新規プロジェクトの作成時に **NetworkPolicy** オブジェクトを自動的に含めるように新規プロジェクトテンプレートを変更できます。

### 3.1.8. 新規プロジェクトのテンプレートの変更

クラスター管理者は、デフォルトのプロジェクトテンプレートを変更し、新規プロジェクトをカスタム要件に基づいて作成することができます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

#### 手順

1. **cluster-admin** 権限を持つユーザーとしてログインしている。
2. デフォルトのプロジェクトテンプレートを生成します。

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで生成される **template.yaml** ファイルを変更します。
4. プロジェクトテンプレートは、**openshift-config** namespace に作成される必要があります。変更したテンプレートを読み込みます。

```
$ oc create -f template.yaml -n openshift-config
```

5. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。

- Web コンソールの使用
  - i. **Administration** → **Cluster Settings** ページに移動します。
  - ii. **Global Configuration** をクリックし、すべての設定リソースを表示します。
  - iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。
- CLI の使用
  - i. **project.config.openshift.io/cluster** リソースを編集します。

```
$ oc edit project.config.openshift.io/cluster
```

6. **spec** セクションを、**projectRequestTemplate** および **name** パラメーターを組み込むように更新し、アップロードされたプロジェクトテンプレートの名前を設定します。デフォルト名は **project-request** です。

## カスタムプロジェクトテンプレートを含むプロジェクト設定リソース

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

- 変更を保存した後、変更が正常に適用されたことを確認するために、新しいプロジェクトを作成します。

### 3.1.8.1. 新規プロジェクトへのネットワークポリシーの追加

クラスター管理者は、ネットワークポリシーを新規プロジェクトのデフォルトテンプレートに追加できます。OpenShift Container Platform は、プロジェクトのテンプレートに指定されたすべての **NetworkPolicy** オブジェクトを自動的に作成します。

#### 前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用する (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- cluster-admin** 権限を持つユーザーとしてクラスターにログインすること。
- 新規プロジェクトのカスタムデフォルトプロジェクトテンプレートを作成していること。

#### 手順

- 以下のコマンドを実行して、新規プロジェクトのデフォルトテンプレートを編集します。

```
$ oc edit template <project_template> -n openshift-config
```

**<project\_template>** を、クラスターに設定したデフォルトテンプレートの名前に置き換えます。デフォルトのテンプレート名は **project-request** です。

- テンプレートでは、各 **NetworkPolicy** オブジェクトを要素として **objects** パラメーターに追加します。**objects** パラメーターは、1つ以上のオブジェクトのコレクションを受け入れます。以下の例では、**objects** パラメーターのコレクションにいくつかの **NetworkPolicy** オブジェクトが含まれます。

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector:
      ingress:
        - from:
```

```

- podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                network.openshift.io/policy-group: ingress
        podSelector: {}
    policyTypes:
      - Ingress
...

```

3. オプション: 以下のコマンドを実行して、新規プロジェクトを作成し、ネットワークポリシーオブジェクトが正常に作成されることを確認します。

- a. 新規プロジェクトを作成します。

```
$ oc new-project <project> ❶
```

❶ **<project>** を、作成しているプロジェクトの名前に置き換えます。

- b. 新規プロジェクトテンプレートのネットワークポリシーオブジェクトが新規プロジェクトに存在することを確認します。

```

$ oc get networkpolicy
NAME                                POD-SELECTOR  AGE
allow-from-openshift-ingress        <none>        7s
allow-from-same-namespace           <none>        7s

```

## 3.2. DNS をプライベートに設定する

クラスターのデプロイ後に、プライベートゾーンのみを使用するように DNS を変更できます。

### 手順

1. クラスターの **DNS** カスタムリソースを確認します。

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

### 出力例

```

apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster

```

```
uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructureID>-int
      kubernetes.io/cluster/<infrastructureID>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
  status: {}
```

**spec** セクションには、プライベートゾーンとパブリックゾーンの両方が含まれることに注意してください。

2. **DNS** カスタムリソースにパッチを適用して、パブリックゾーンを削除します。

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec": {"publicZone": null}}'
dns.config.openshift.io/cluster patched
```

Ingress コントローラーは **Ingress** オブジェクトの作成時に **DNS** 定義を参照するため、**Ingress** オブジェクトを作成または変更する場合、プライベートレコードのみが作成されます。



### 重要

既存の Ingress オブジェクトの DNS レコードは、パブリックゾーンの削除時に変更されません。

3. オプション: クラスターの **DNS** カスタムリソースを確認し、パブリックゾーンが削除されていることを確認します。

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

### 出力例

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructureID>-int
      kubernetes.io/cluster/<infrastructureID>-wfp4: owned
  status: {}
```



### 3.3. クラスター全体のプロキシの有効化

プロキシオブジェクトは、クラスター全体の egress プロキシを管理するために使用されます。プロキシを設定せずにクラスターがインストールまたはアップグレードされると、プロキシオブジェクトは引き続き生成されますが、**spec** は設定されません。以下に例を示します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

クラスター管理者は、この **cluster** プロキシオブジェクトを変更して OpenShift Container Platform のプロキシを設定できます。



#### 注記

**cluster** という名前のプロキシオブジェクトのみがサポートされ、追加のプロキシは作成できません。

#### 前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

#### 手順

1. HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる ConfigMap を作成します。



#### 注記

プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

- a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

- 1** このデータキーは **ca-bundle.crt** という名前にする必要があります。
- 2** プロキシのアイデンティティ証明書に署名するために使用される 1 つ以上の PEM でエンコードされた X.509 証明書。

- 3 プロキシオブジェクトから参照される ConfigMap 名。
- 4 ConfigMap は **openshift-config** namespace になければなりません。

b. このファイルから ConfigMap を作成します。

```
$ oc create -f user-ca-bundle.yaml
```

2. **oc edit** コマンドを使用してプロキシオブジェクトを変更します。

```
$ oc edit proxy/cluster
```

3. プロキシに必要なフィールドを設定します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
  readinessEndpoints:
    - http://www.google.com 4
    - https://www.google.com
  trustedCA:
    name: user-ca-bundle 5
```

- 1 クラスター外の HTTP 接続を作成するために使用するプロキシ URL。URL スキームは **http** である必要があります。
- 2 クラスター外で HTTPS 接続を作成するために使用するプロキシ URL。これが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。
- 3 プロキシを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。

サブドメインのみと一致するように、ドメインの前に . を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、**y.com** には一致しません。\* を使用し、すべての宛先のプロキシをバイパスします。インストール設定で **networking.machineNetwork[].cidr** フィールドで定義されるネットワークに含まれていないワーカーをスケールアップする場合、それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

**httpProxy** または **httpsProxy** フィールドのいずれも設定されていない場合に、このフィールドは無視されます。

- 4 **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。
- 5 HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の ConfigMap の参照。ここで参照する前に ConfigMap が存在している必要があります。このフィールドは、プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

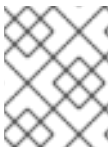


#### 注記

URL スキームは **http** である必要があります。**https** スキームは現在サポートされていません。

### 3.4. CLUSTER NETWORK OPERATOR (CNO) の設定

クラスターネットワークの設定は、Cluster Network Operator (CNO) 設定の一部として指定され、**cluster** という名前の CR オブジェクトに保存されます。CR は **operator.openshift.io** API グループの **Network** API のパラメーターを指定します。



#### 注記

クラスターのインストール後に、クラスターネットワークプロバイダーの設定を変更することはできません。

### 3.5. INGRESS クラスタートラフィックの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする以下の方法を提供します。

- HTTP/HTTPS を使用する場合は Ingress コントローラーを使用する。
- HTTPS 以外の TLS で暗号化されたプロトコルを使用する場合 (TLS と SNI ヘッダーの使用など) は Ingress コントローラーを使用する。
- それ以外の場合は、ロードバランサー、外部 IP、またはノードポートを使用します。

方法	目的
Ingress コントローラーの使用	HTTP/HTTPS トラフィックおよび HTTPS 以外の TLS で暗号化されたプロトコル (TLS と SNI ヘッダーの使用など) へのアクセスを許可します。
ロードバランサーサービスを使用した外部 IP の自動割り当て	プールから割り当てられた IP アドレスを使った非標準ポートへのトラフィックを許可します。
外部 IP のサービスへの手動割り当て	特定の IP アドレスを使った非標準ポートへのトラフィックを許可します。
NodePort の設定	クラスターのすべてのノードでサービスを公開します。

### 3.6. RED HAT OPENSIFT SERVICE MESH でサポートされている設定

以下は、Red Hat OpenShift Service Mesh で唯一サポートされている設定です。

- Red Hat OpenShift Container Platform バージョン 4.x。



## 注記

OpenShift Online および OpenShift Dedicated は Red Hat OpenShift Service Mesh に対してはサポートされていません。

- デプロイメントは、フェデレーションされていない単一の OpenShift Container Platform クラスターに含まれる必要があります。
- Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform x86\_64 のみ利用できます。
- 本リリースでは、すべてのサービスメッシュコンポーネントが OpenShift クラスターに含まれ、動作している設定のみをサポートしています。クラスター外にあるマイクロサービスの管理や、マルチクラスターシナリオにおけるマイクロサービスの管理はサポートしていません。
- 本リリースでは、仮想マシンなどの外部サービスを統合していない設定のみをサポートしています。

### 3.6.1. Red Hat OpenShift Service Mesh でサポートされている Kiali の設定

- Kiali の可観測性コンソールは Chrome、Edge、Firefox、または Safari ブラウザーの 2 つの最新リリースでのみサポートされています。

### 3.6.2. サポートされている Mixer アダプター

- 本リリースでは、次の Mixer アダプターのみをサポートしています。
  - 3scale Istio Adapter

### 3.6.3. Red Hat OpenShift Service Mesh のインストールアクティビティー

Red Hat OpenShift Service Mesh Operator をインストールするには、まず以下の Operator をインストールする必要があります。

- **Elasticsearch**: オープンソースの [Elasticsearch](#) プロジェクトをベースとし、Jaeger を使用してトレースとロギングを行うために Elasticsearch クラスターを設定し、管理することができます。
- **Jaeger**: オープンソース [Jaeger](#) プロジェクトをベースとし、トレースを実行して、複雑な分散システムでトランザクションを監視し、トラブルシューティングできます。
- **Kiali**: オープンソースの [Kiali](#) プロジェクトをベースとしており、サービスメッシュの可観測性を提供します。Kiali を使用すると、単一のコンソールで設定を表示し、トラフィックを監視し、トレースの表示と分析を実行できます。

Elasticsearch、Jaeger、Kiali Operator のインストール後に、Red Hat OpenShift Service Mesh Operator をインストールします。Service Mesh Operator は、サービスメッシュコンポーネントのデプロイメント、更新、および削除を管理する **ServiceMeshControlPlane** リソースを定義し、監視します。

- **Red Hat OpenShift Service Mesh** オープンソースの [Istio](#) プロジェクトに基づき、アプリケーションを設定するマイクロサービスを接続し、保護し、制御し、観察することができます。

## 次のステップ

- OpenShift Container Platform 環境に [Red Hat OpenShift Service Mesh](#) をインストール します。

## 3.7. 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。

### 3.7.1. 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
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。

### 3.7.2. 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4章 インストール後のストレージ設定

OpenShift Container Platform のインストール後に、ストレージの設定を含め、クラスターをさらに拡張し、要件に合わせてカスタマイズできます。

### 4.1. 動的プロビジョニング

#### 4.1.1. 動的プロビジョニングについて

**StorageClass** リソースオブジェクトは、要求可能なストレージを記述し、分類するほか、動的にプロビジョニングされるストレージのパラメーターを要求に応じて渡すための手段を提供します。**StorageClass** オブジェクトは、さまざまなレベルのストレージとストレージへのアクセスを制御するための管理メカニズムとしても機能します。クラスター管理者 (**cluster-admin**) またはストレージ管理者 (**storage-admin**) は、ユーザーが基礎となるストレージボリュームソースに関する詳しい知識がなくても要求できる **StorageClass** オブジェクトを定義し、作成します。

OpenShift Container Platform の永続ボリュームフレームワークはこの機能を有効にし、管理者がクラスターに永続ストレージをプロビジョニングできるようにします。フレームワークにより、ユーザーは基礎となるインフラストラクチャーの知識がなくてもこれらのリソースを要求できるようになります。

OpenShift Container Platform では、数多くのストレージタイプを永続ボリュームとして使用することができます。これらはすべて管理者によって静的にプロビジョニングされますが、一部のストレージタイプは組み込みプロバイダーとプラグイン API を使用して動的に作成できます。

#### 4.1.2. 利用可能な動的プロビジョニングプラグイン

OpenShift Container Platform は、以下のプロビジョナープラグインを提供します。これらには、クラスターの設定済みプロバイダーの API を使用して新規ストレージリソースを作成する動的プロビジョニング用の一般的な実装が含まれます。

ストレージタイプ	プロビジョナープラグインの名前	注記
Red Hat OpenStack Platform (RHOSP) Cinder	<b>kubernetes.io/cinder</b>	
RHOSP Manila Container Storage Interface (CSI)	<b>manila.csi.openstack.org</b>	インストールが完了すると、OpenStack Manila CSI Driver Operator および ManilaDriver は、動的プロビジョニングに必要なすべての利用可能な Manila 共有タイプに必要なストレージクラスを自動的に作成します。
AWS Elastic Block Store (EBS)	<b>kubernetes.io/aws-ebs</b>	複数クラスターを複数の異なるゾーンで使用する際の動的プロビジョニングの場合、各ノードに <b>Key=kubernetes.io/cluster/&lt;cluster_name&gt;,Value=&lt;cluster_id&gt;</b> のタグを付けます。ここで、 <b>&lt;cluster_name&gt;</b> および <b>&lt;cluster_id&gt;</b> はクラスターごとに固有の値になります。

ストレージタイプ	プロビジョナープラグインの名前	注記
Azure Disk	<b>kubernetes.io/azure-disk</b>	
Azure File	<b>kubernetes.io/azure-file</b>	<b>persistent-volume-binder</b> サービスアカウントでは、Azure ストレージアカウントおよびキーを保存するためにシークレットを作成し、取得するためのパーミッションが必要です。
GCE Persistent Disk (gcePD)	<b>kubernetes.io/gce-pd</b>	マルチゾーン設定では、GCE プロジェクトごとに OpenShift Container Platform クラスターを実行し、現行クラスターのノードが存在しないゾーンで PV が作成されないようにすることが推奨されます。
VMware vSphere	<b>kubernetes.io/vsphere-volume</b>	



### 重要

選択したプロビジョナープラグインでは、関連するクラウド、ホスト、またはサードパーティープロバイダーを、関連するドキュメントに従って設定する必要があります。

## 4.2. ストレージクラスの定義

現時点で、**StorageClass** オブジェクトはグローバルスコープオブジェクトであり、**cluster-admin** または **storage-admin** ユーザーによって作成される必要があります。



### 重要

Cluster Storage Operator は、使用されるプラットフォームに応じてデフォルトのストレージクラスをインストールする可能性があります。このストレージクラスは Operator によって所有され、制御されます。アノテーションとラベルを定義するほかは、これを削除したり、変更したりすることはできません。異なる動作が必要な場合は、カスタムストレージクラスを定義する必要があります。

以下のセクションでは、**StorageClass** オブジェクトの基本的な定義とサポートされている各プラグインタイプの具体的な例について説明します。

### 4.2.1. 基本 StorageClass オブジェクト定義

以下のリソースは、ストレージクラスを設定するために使用するパラメーターおよびデフォルト値を示しています。この例では、AWS ElasticBlockStore (EBS) オブジェクト定義を使用します。

#### StorageClass 定義の例



```
kind: StorageClass ❶
apiVersion: storage.k8s.io/v1 ❷
metadata:
  name: gp2 ❸
  annotations: ❹
    storageclass.kubernetes.io/is-default-class: 'true'
  ...
provisioner: kubernetes.io/aws-ebs ❺
parameters: ❻
  type: gp2
...
```

- ❶ (必須) API オブジェクトタイプ。
- ❷ (必須) 現在の apiVersion。
- ❸ (必須) ストレージクラスの名前。
- ❹ (オプション) ストレージクラスのアノテーション。
- ❺ (必須) このストレージクラスに関連付けられているプロビジョナーのタイプ。
- ❻ (オプション) 特定のプロビジョナーに必要なパラメーター。これはプラグインによって異なります。

#### 4.2.2. ストレージクラスのアノテーション

ストレージクラスをクラスター全体のデフォルトとして設定するには、以下のアノテーションをストレージクラスのメタデータに追加します。

```
storageclass.kubernetes.io/is-default-class: "true"
```

以下に例を示します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

これにより、特定のストレージクラスを指定しない永続ボリューム要求 (PVC) がデフォルトのストレージクラスによって自動的にプロビジョニングされるようになります。



#### 注記

ベータアノテーションの **storageclass.beta.kubernetes.io/is-default-class** は依然として使用可能ですが、今後のリリースで削除される予定です。

ストレージクラスの記述を設定するには、以下のアノテーションをストレージクラスのメタデータに追加します。

kubernetes.io/description: My Storage Class Description

以下に例を示します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

### 4.2.3. RHOSP Cinder オブジェクトの定義

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: kubernetes.io/cinder
parameters:
  type: fast ①
  availability: nova ②
  fsType: ext4 ③
```

- ① Cinder で作成されるボリュームタイプ。デフォルトは空です。
- ② アベイラビリティゾーン。指定しない場合、ボリュームは通常 OpenShift Container Platform クラスターのノードがあるすべてのアクティブゾーンでラウンドロビンされます。
- ③ 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

### 4.2.4. AWS Elastic Block Store (EBS) オブジェクト定義

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 ①
  iopsPerGB: "10" ②
  encrypted: "true" ③
  kmsKeyId: keyvalue ④
  fsType: ext4 ⑤
```

- ① (必須) **io1**、**gp2**、**sc1**、**st1** から選択します。デフォルトは **gp2** です。有効な Amazon Resource Names (ARN) 値については、[AWS のドキュメント](#) を参照してください。

Name (ARN) 値については、[AWS のドキュメント](#) を参照してください。

- ② (オプション) io1 ボリュームのみ。1 GiB あたり 1 秒あたりの I/O 処理数。AWS ボリュームプラグインは、この値と要求されたボリュームのサイズを乗算してボリュームの IOPS を算出します。値の上限は、AWS でサポートされる最大値である 20,000 IOPS です。詳細については、[AWS のドキュメント](#) を参照してください。
- ③ (オプション) EBS ボリュームを暗号化するかどうかを示します。有効な値は **true** または **false** です。
- ④ (オプション) ボリュームを暗号化する際に使用するキーの完全な ARN。値を指定しない場合でも **encrypted** が **true** に設定されている場合は、AWS によってキーが生成されます。有効な ARN 値については、[AWS のドキュメント](#) を参照してください。
- ⑤ (オプション) 動的にプロビジョニングされたボリュームで作成されるファイルシステム。この値は、動的にプロビジョニングされる永続ボリュームの **fsType** フィールドにコピーされ、ボリュームの初回マウント時にファイルシステムが作成されます。デフォルト値は **ext4** です。

#### 4.2.5. Azure Disk オブジェクト定義

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-premium
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer ①
allowVolumeExpansion: true
parameters:
  kind: Managed ②
  storageaccounttype: Premium_LRS ③
reclaimPolicy: Delete
```

- ① **WaitForFirstConsumer** を使用することが強く推奨されます。これにより、Pod を利用可能なゾーンから空きのあるワーカーノードにスケジュールするのに十分なストレージがボリュームプロビジョニングされます。
- ② 許容値は、**Shared** (デフォルト)、**Managed**、および **Dedicated** です。



#### 重要

Red Hat は、ストレージクラスでの **kind: Managed** の使用のみをサポートします。

**Shared** および **Dedicated** の場合、Azure は管理対象外のディスクを作成しますが、OpenShift Container Platform はマシンの OS (root) ディスクの管理ディスクを作成します。ただし、Azure Disk はノードで管理ディスクおよび管理対象外ディスクの両方の使用を許可しないため、**Shared** または **Dedicated** で作成された管理対象外ディスクを OpenShift Container Platform ノードに割り当てることはできません。

3

Azure ストレージアカウントの SKU の層。デフォルトは空です。プレミアム VM は **Standard\_LRS** ディスクと **Premium\_LRS** ディスクの両方を割り当て、標準 VM は

- a. **kind** が **Shared** に設定されている場合は、Azure は、クラスターと同じリソースグループにあるいくつかの共有ストレージアカウントで、アンマネージドディスクをすべて作成します。
- b. **kind** が **Managed** に設定されている場合は、Azure は新しいマネージドディスクを作成します。
- c. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されている場合には、Azure は、クラスターと同じリソースグループ内にある新規のアンマネージドディスク用に、指定のストレージアカウントを使用します。これを機能させるには、以下が前提となります。
  - 指定のストレージアカウントが、同じリージョン内にあること。
  - Azure Cloud Provider にストレージアカウントへの書き込み権限があること。
- d. **kind** が **Dedicated** に設定されており、**storageAccount** が指定されていない場合には、Azure はクラスターと同じリソースグループ内の新規のアンマネージドディスク用に、新しい専用のストレージアカウントを作成します。

#### 4.2.6. Azure File のオブジェクト定義

Azure File ストレージクラスはシークレットを使用して Azure ストレージアカウント名と Azure ファイル共有の作成に必要なストレージアカウントキーを保存します。これらのパーミッションは、以下の手順の一部として作成されます。

##### 手順

1. シークレットの作成および表示を可能にする **ClusterRole** オブジェクトを定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: [""]
  resources: ['secrets']
  verbs: ['get','create']
```

- 1 シークレットを表示し、作成するためのクラスターロールの名前。

2. クラスターロールをサービスアカウントに追加します。

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder>
```

##### 出力例

```
system:serviceaccount:kube-system:persistent-volume-binder
```

3. Azure File **StorageClass** オブジェクトを作成します。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> ❶
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus ❷
  skuName: Standard_LRS ❸
  storageAccount: <storage-account> ❹
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

- ❶ ストレージクラス名永続ボリューム要求 (PVC) は、関連する永続ボリュームをプロビジョニングするためにこのストレージクラスを使用します。
- ❷ **eastus** などの Azure ストレージアカウントの場所。デフォルトは空であり、新規 Azure ストレージアカウントが OpenShift Container Platform クラスターの場所に作成されず。
- ❸ SKU は、**Standard\_LRS** などの Azure ストレージアカウントの層になります。デフォルトは空です。つまり、新しい Azure ストレージアカウントは **Standard\_LRS** SKU で作成されます。
- ❹ Azure ストレージアカウントの名前。ストレージアカウントが提供されると、**skuName** および **location** は無視されます。ストレージアカウントを指定しない場合、ストレージクラスは、定義された **skuName** および **location** に一致するアカウントのリソースグループに関連付けられたストレージアカウントを検索します。

## 4.2.6.1. Azure File を使用する場合の考慮事項

以下のファイルシステム機能は、デフォルトの Azure File ストレージクラスではサポートされません。

- シンボリックリンク
- ハードリンク
- 拡張属性
- スパースファイル
- 名前付きパイプ

また、Azure File がマウントされるディレクトリーの所有者 ID (UID) は、コンテナのプロセス UID とは異なります。**uid** マウントオプションは **StorageClass** オブジェクトに指定して、マウントされたディレクトリーに使用する特定のユーザー ID を定義できます。

以下の **StorageClass** オブジェクトは、マウントされたディレクトリーのシンボリックリンクを有効にした状態で、ユーザーおよびグループ ID を変更する方法を示しています。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```

  name: azure-file
  mountOptions:
    - uid=1500 ❶
    - gid=1500 ❷
    - mfsymlinks ❸
  provisioner: kubernetes.io/azure-file
  parameters:
    location: eastus
    skuName: Standard_LRS
    reclaimPolicy: Delete
    volumeBindingMode: Immediate

```

❶ マウントされたディレクトリーに使用するユーザー ID を指定します。

❷ マウントされたディレクトリーに使用するグループ ID を指定します。

❸ シンボリックリンクを有効にします。

#### 4.2.7. GCE PersistentDisk (gcePD) オブジェクトの定義

##### gce-pd-storageclass.yaml

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard ❶
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete

```

❶ **pd-standard** または **pd-ssd** のいずれかを選択します。デフォルトは **pd-standard** です。

#### 4.2.8. VMWare vSphere オブジェクトの定義

##### vsphere-storageclass.yaml

```

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/vsphere-volume ❶
parameters:
  diskformat: thin ❷

```

- 1 OpenShift Container Platform で VMware vSphere を使用方法の詳細については、[VMware vSphere のドキュメント](#) を参照してください。
- 2 **diskformat: thin**、**zeroedthick** および **eagerzeroedthick** はすべて有効なディスクフォーマットです。ディスクフォーマットの種類に関する詳細は、vSphere のドキュメントを参照してください。デフォルト値は **thin** です。

### 4.3. デフォルトストレージクラスの変更

AWS を使用している場合は、以下のプロセスを使用してデフォルトのストレージクラスを変更します。このプロセスでは、**gp2** と **standard** の2つのストレージクラスが定義されており、デフォルトのストレージクラスを **gp2** から **standard** に変更する必要がある場合を想定しています。

1. ストレージクラスを一覧表示します。

```
$ oc get storageclass
```

#### 出力例

NAME	TYPE
gp2 (default)	kubernetes.io/aws-ebs 1
standard	kubernetes.io/aws-ebs

- 1 **(default)** はデフォルトのストレージクラスを示します。

2. デフォルトのストレージクラスのアノテーション **storageclass.kubernetes.io/is-default-class** の値を **false** に変更します。

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. アノテーションを追加するか、またはアノテーションを **storageclass.kubernetes.io/is-default-class=true** として変更することで、別のストレージクラスをデフォルトにします。

```
$ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

4. 変更内容を確認します。

```
$ oc get storageclass
```

#### 出力例

NAME	TYPE
gp2	kubernetes.io/aws-ebs
standard (default)	kubernetes.io/aws-ebs

### 4.4. 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。

## 4.5. 利用可能な永続ストレージオプション

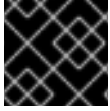
永続ストレージオプションについて理解し、OpenShift Container Platform 環境を最適化できるようにします。

表4.1 利用可能なストレージオプション

ストレージタイプ	説明	例
ブロック	<ul style="list-style-type: none"> <li>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。</li> <li>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。</li> <li>ストレージエリアネットワーク (SAN) と呼ばれます。</li> <li>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。</li> </ul>	AWS EBS および VMware vSphere は、OpenShift Container Platform で永続ボリューム (PV) の動的なプロビジョニングをサポートします。
ファイル	<ul style="list-style-type: none"> <li>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。</li> <li>ネットワークアタッチストレージ (NAS) と呼ばれます。</li> <li>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。</li> </ul>	RHEL NFS、NetApp NFS <sup>[1]</sup> 、および Vendor NFS
オブジェクト	<ul style="list-style-type: none"> <li>REST API エンドポイント経由でアクセスできます。</li> <li>OpenShift Container Platform レジストリーで使用するために設定できます。</li> <li>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に組み込む必要があります。</li> </ul>	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。

## 4.6. 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4.2 設定可能な推奨ストレージ技術

ストレージタイプ	ROX <sup>1</sup>	RWX <sup>2</sup>	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 <sup>3</sup>	ロギング	アプリ
ブロック	はい <sup>4</sup>	いいえ	設定可能	設定不可	推奨	推奨	推奨
ファイル	はい <sup>4</sup>	○	設定可能	設定可能	設定可能 <sup>5</sup>	設定可能 <sup>6</sup>	推奨
オブジェクト	○	○	推奨	推奨	設定不可	設定不可	設定不可 <sup>7</sup>

<sup>1</sup> **ReadOnlyMany**

<sup>2</sup> **ReadWriteMany**

<sup>3</sup> 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。

<sup>4</sup> 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。

<sup>5</sup> 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。

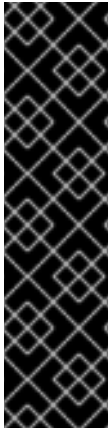
<sup>6</sup> ロギングの場合、共有ストレージを使用することはアンチパターンとなります。elasticsearch ごとに1つのボリュームが必要です。

<sup>7</sup> 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。

### 4.6.1. 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、およびログGING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する 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。

### 4.6.1.1. 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。

### 4.6.1.2. 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。

### 4.6.1.3. 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。



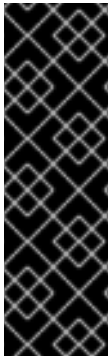
### 重要

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。

#### 4.6.1.4. ロギング

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

- 推奨されるストレージ技術はブロックストレージです。
- 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する 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。

#### 4.6.1.5. 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。

#### 4.6.2. 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傾向にあります。

## 4.7. RED HAT OPENSIFT CONTAINER STORAGE のデプロイ

Red Hat OpenShift Container Storage は、インハウスまたはハイブリッドクラウドのいずれの場合でもファイル、ブロックおよびオブジェクトストレージをサポートし、OpenShift Container Platform のすべてに対応する永続ストレージのプロバイダーです。Red Hat のストレージソリューションとして、Red Hat OpenShift Container Storage は、デプロイメント、管理およびモニタリングを行うために OpenShift Container Platform に完全に統合されています。

Red Hat OpenShift Container Storage についてのトピック	Red Hat OpenShift Container Storage ドキュメントの参照先
新機能、既知の問題、主なバグ修正およびテクノロジープレビュー	<a href="#">OpenShift Container Storage 4.5 Release Notes</a>
サポートされるワークロード、レイアウト、ハードウェアおよびソフトウェア要件、サイジング、スケールリングに関する推奨事項	<a href="#">Planning your OpenShift Container Storage 4.5 deployment</a>
環境がインターネットに直接接続していない場合のデプロイ準備手順	<a href="#">Preparing to deploy OpenShift Container Storage 4.5 in a disconnected environment</a>
外部の Red Hat Ceph Storage クラスターを使用するように OpenShift Container Storage をデプロイする手順	<a href="#">Deploying OpenShift Container Storage 4.5 in external mode</a>
ベアメタルインフラストラクチャーでの OpenShift Container Storage のローカルストレージへのデプロイ手順	<a href="#">ベアメタルインフラストラクチャーを使用した OpenShift Container Storage 4.5 のデプロイ</a>
Red Hat OpenShift Container Platform VMWare vSphere クラスターへの OpenShift Container Storage のデプロイ手順	<a href="#">VMWare vSphere への OpenShift Container Storage 4.5 のデプロイ</a>
ローカルまたはクラウドストレージの Amazon Web Services を使用した OpenShift Container Storage のデプロイ手順	<a href="#">Amazon Web Services を使用した OpenShift Container Storage 4.5 のデプロイ</a>
既存の Red Hat OpenShift Container Platform Google Cloud クラスターへの OpenShift Container Storage のデプロイおよび管理手順	<a href="#">Google Cloud を使用した OpenShift Container Storage 4.5 のデプロイおよび管理</a>
既存の Red Hat OpenShift Container Platform Azure クラスターへの OpenShift Container Storage のデプロイおよび管理手順	<a href="#">Microsoft Azure を使用した OpenShift Container Storage 4.5 のデプロイおよび管理</a>
Red Hat OpenShift Container Storage 4.5 クラスターの管理	<a href="#">Managing OpenShift Container Storage 4.5</a>

Red Hat OpenShift Container Storage についてのトピック	Red Hat OpenShift Container Storage ドキュメントの参照先
Red Hat OpenShift Container Storage 4.5 クラスターのモニタリング	<a href="#">Monitoring Red Hat OpenShift Container Storage 4.5</a>
操作中に発生する問題の解決	<a href="#">Troubleshooting OpenShift Container Storage 4.5</a>
OpenShift Container Platform クラスターのバージョン 3 からバージョン 4 への移行	<a href="#">移行</a>

## 第5章 ユーザー向けの準備

OpenShift Container Platform のインストール後に、ユーザー向けに準備するための手順を含め、クラスターをさらに拡張し、要件に合わせてカスタマイズできます。

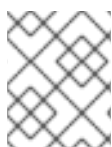
### 5.1. アイデンティティプロバイダー設定について

OpenShift Container Platform コントロールプレーンには、組み込まれた OAuth サーバーが含まれます。開発者および管理者は OAuth アクセストークンを取得して、API に対して認証します。

管理者は、クラスターのインストール後に、OAuth をアイデンティティプロバイダーを指定するように設定できます。

#### 5.1.1. OpenShift Container Platform のアイデンティティプロバイダーについて

デフォルトでは、**kubeadmin** ユーザーのみがクラスターに存在します。アイデンティティプロバイダーを指定するには、アイデンティティプロバイダーを記述し、これをクラスターに追加するカスタムリソースを作成する必要があります。



#### 注記

/, :, および % を含む OpenShift Container Platform ユーザー名はサポートされません。

#### 5.1.2. サポートされるアイデンティティプロバイダー

以下の種類のアイデンティティプロバイダーを設定できます。

アイデンティティプロバイダー	説明
HTPasswd	<b>htpasswd</b> アイデンティティプロバイダーを <b>htpasswd</b> を使用して生成されたフラットファイルに対してユーザー名とパスワードを検証するように設定します。
Keystone	<b>keystone</b> アイデンティティプロバイダーを、OpenShift Container Platform クラスターを Keystone に統合し、ユーザーを内部データベースに保存するように設定された OpenStack Keystone v3 サーバーによる共有認証を有効にするように設定します。
LDAP	<b>ldap</b> アイデンティティプロバイダーを、単純なバインド認証を使用して LDAPv3 サーバーに対してユーザー名とパスワードを検証するように設定します。
Basic 認証	<b>basic-authentication</b> アイデンティティプロバイダーを、ユーザーがリモートアイデンティティプロバイダーに対して検証された認証情報を使って OpenShift Container Platform にログインできるように設定します。Basic 認証は、汎用的なバックエンド統合メカニズムです。
要求ヘッダー	<b>request-header</b> アイデンティティプロバイダーを、 <b>X-Remote-User</b> などの要求ヘッダー値から識別するように設定します。通常、これは要求ヘッダー値を設定する認証プロキシと併用されます。

アイデンティティプロバイダー	説明
<a href="#">GitHub または GitHub Enterprise</a>	<b>github</b> アイデンティティプロバイダーを、GitHub または GitHub Enterprise の OAuth 認証サーバーに対してユーザー名とパスワードを検証するように設定します。
<a href="#">GitLab</a>	<b>gitlab</b> アイデンティティプロバイダーを、 <a href="#">GitLab.com</a> またはその他の GitLab インスタンスをアイデンティティプロバイダーとして使用するよう設定します。
<a href="#">Google</a>	<b>google</b> アイデンティティプロバイダーを、 <a href="#">Google の OpenID Connect 統合</a> を使用して設定します。
<a href="#">OpenID Connect</a>	<b>oidc</b> アイデンティティプロバイダーを、 <a href="#">Authorization Code Flow</a> を使用して OpenID Connect アイデンティティプロバイダーと統合するよう設定します。

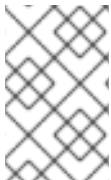
アイデンティティプロバイダーを定義した後に、[RBAC を使用してパーミッションの定義および適用](#)を実行できます。

### 5.1.3. アイデンティティプロバイダーパラメーター

以下のパラメーターは、すべてのアイデンティティプロバイダーに共通するパラメーターです。

パラメーター	説明
<b>name</b>	プロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。

パラメーター	説明
<b>mappingMethod</b>	<p>新規アイデンティティーがログイン時にユーザーにマップされる方法を定義します。以下の値のいずれかを入力します。</p> <p><b>claim</b></p> <p>デフォルトの値です。アイデンティティーの推奨ユーザー名を持つユーザーをプロビジョニングします。そのユーザー名を持つユーザーがすでに別のアイデンティティーにマッピングされている場合は失敗します。</p> <p><b>lookup</b></p> <p>既存のアイデンティティー、ユーザーアイデンティティーマッピング、およびユーザーを検索しますが、ユーザーまたはアイデンティティーの自動プロビジョニングは行いません。これにより、クラスター管理者は手動で、または外部のプロセスを使用してアイデンティティーとユーザーを設定できます。この方法を使用する場合は、ユーザーを手動でプロビジョニングする必要があります。</p> <p><b>generate</b></p> <p>アイデンティティーの推奨ユーザー名を持つユーザーをプロビジョニングします。推奨ユーザー名を持つユーザーがすでに既存のアイデンティティーにマッピングされている場合は、一意のユーザー名が生成されます。例: <b>myuser2</b>この方法は、OpenShift Container Platform のユーザー名とアイデンティティープロバイダーのユーザー名との正確な一致を必要とする外部プロセス (LDAP グループ同期など) と組み合わせて使用することはできません。</p> <p><b>add</b></p> <p>アイデンティティーの推奨ユーザー名を持つユーザーをプロビジョニングします。推奨ユーザー名を持つユーザーがすでに存在する場合、アイデンティティーは既存のユーザーにマッピングされ、そのユーザーの既存のアイデンティティーマッピングに追加されます。これは、同じユーザーセットを識別して同じユーザー名にマッピングするアイデンティティープロバイダーが複数設定されている場合に必要です。</p>



## 注記

**mappingMethod** パラメーターを **add** に設定すると、アイデンティティープロバイダーの追加または変更時に新規プロバイダーのアイデンティティーを既存ユーザーにマッピングできます。

### 5.1.4. アイデンティティープロバイダー CR のサンプル

以下のカスタムリソース (CR) は、アイデンティティープロバイダーを設定するために使用するパラメーターおよびデフォルト値を示します。この例では、HTPasswd アイデンティティープロバイダーを使用しています。

#### アイデンティティープロバイダー CR のサンプル

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
    - name: my_identity_provider ❶
      mappingMethod: claim ❷
      type: HTPasswd
```



```
htpasswd:
  fileData:
    name: htpass-secret 3
```

- 1 このプロバイダー名は、プロバイダーのユーザー名に接頭辞として付加され、アイデンティティ名が作成されます。
- 2 このプロバイダーのアイデンティティと **User** オブジェクト間にマッピングが確立される方法を制御します。
- 3 **htpasswd** を使用して生成されたファイルが含まれる既存のシークレットです。

## 5.2. RBAC の使用によるパーミッションの定義および適用

ロールベースのアクセス制御について理解し、これを適用します。

### 5.2.1. RBAC の概要

Role-based Access Control (RBAC: ロールベースアクセス制御) オブジェクトは、ユーザーがプロジェクト内で所定のアクションを実行することが許可されるかどうかを決定します。

これにより、プラットフォーム管理者はクラスターロールおよびバインディングを使用して、OpenShift Container Platform プラットフォーム自体およびすべてのプロジェクトへの各種のアクセスレベルを持つユーザーを制御できます。

開発者はローカルロールとバインディングを使用して、プロジェクトにアクセスできるユーザーを制御できます。認可、認証とは異なる別の手順であることに注意してください。認可の手順は、アクションを実行するユーザーのアイデンティティの判別により密接に関連しています。

認可は以下を使用して管理されます。

認可オブジェクト	説明
ルール	オブジェクトのセットで許可されている動詞のセット(例: ユーザーまたはサービスアカウントが Pod の <b>create</b> を実行できるかどうか)
ロール	ルールのコレクション。ユーザーおよびグループを複数のロールに関連付けたり、バインドしたりできます。
バインディング	ロールを使ったユーザー/グループ間の関連付けです。

2つのレベルのRBAC ロールおよびバインディングが認可を制御します。

RBAC レベル	説明
クラスター RBAC	すべてのプロジェクトで適用可能なロールおよびバインディングです。 <b>クラスターロール</b> はクラスター全体で存在し、 <b>クラスターロールのバインディング</b> はクラスターロールのみを参照できます。

RBAC レベル	説明
ローカル RBAC	所定のプロジェクトにスコープ設定されているロールおよびバインディングです。 <b>ローカルロール</b> は単一プロジェクトのみに存在し、ローカルロールのバインディングはクラスターロールおよびローカルロールの <b>両方</b> を参照できます。

クラスターのロールバインディングは、クラスターレベルで存在するバインディングですが、ロールバインディングはプロジェクトレベルで存在します。ロールバインディングは、プロジェクトレベルで存在します。クラスターの **view (表示)** ロールは、ユーザーがプロジェクトを表示できるようローカルのロールバインディングを使用してユーザーにバインドする必要があります。ローカルロールは、クラスターのロールが特定の状況に必要なパーミッションのセットを提供しない場合にのみ作成する必要があります。

この2つのレベルからなる階層により、ローカルロールで個別プロジェクト内のカスタマイズが可能になる一方で、クラスターロールによる複数プロジェクト間での再利用が可能になります。

評価時に、クラスターロールのバインディングおよびローカルロールのバインディングが使用されます。以下に例を示します。

1. クラスター全体の allow ルールがチェックされます。
2. ローカルにバインドされた allow ルールがチェックされます。
3. デフォルトで拒否します。

#### 5.2.1.1. デフォルトのクラスターロール

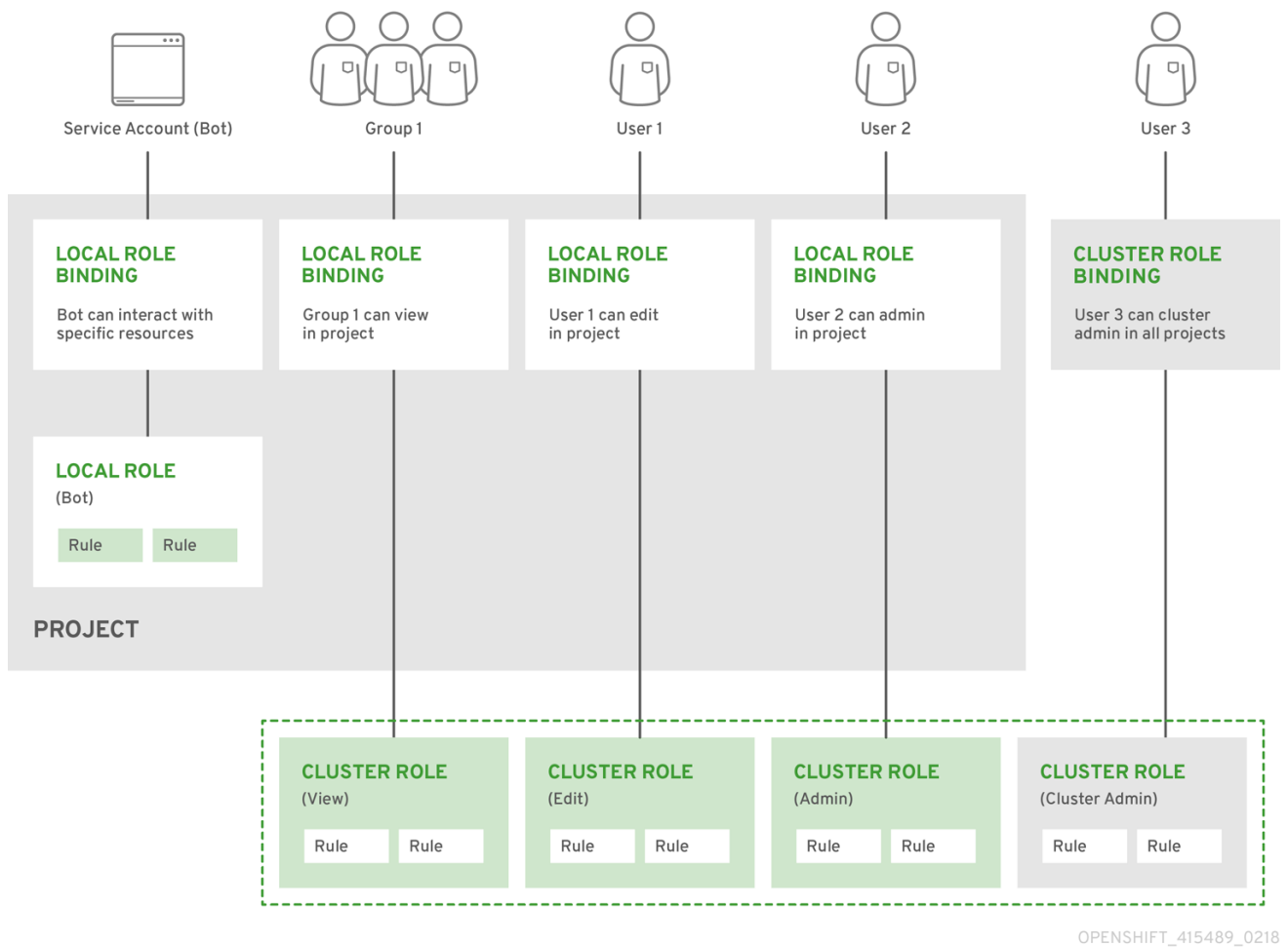
OpenShift Container Platform には、クラスター全体で、またはローカルにユーザーおよびグループにバインドできるデフォルトのクラスターロールのセットが含まれます。必要に応じて、デフォルトのクラスターロールを手動で変更できます。

デフォルトのクラスターロール	説明
<b>admin</b>	プロジェクトマネージャー。ローカルバインディングで 사용되는場合、 <b>admin</b> には、プロジェクト内のすべてのリソースを表示し、クォータ以外のリソース内のすべてのリソースを変更する権限があります。
<b>basic-user</b>	プロジェクトおよびユーザーについての基本的な情報を取得できるユーザーです。
<b>cluster-admin</b>	すべてのプロジェクトですべてのアクションを実行できるスーパーユーザーです。ローカルバインディングでユーザーにバインドされる場合、クォータに対する完全な制御およびプロジェクト内のすべてのリソースに対するすべてのアクションを実行できます。
<b>cluster-status</b>	基本的なクラスターのステータス情報を取得できるユーザーです。

デフォルトのクラスターロール	説明
<b>edit</b>	プロジェクトのほとんどのプロジェクトを変更できるが、ロールまたはバインディングを表示したり、変更したりする機能を持たないユーザーです。
<b>self-provisioner</b>	独自のプロジェクトを作成できるユーザーです。
<b>view</b>	変更できないものの、プロジェクトでほとんどのオブジェクトを確認できるユーザーです。それらはロールまたはバインディングを表示したり、変更したりできません。

ローカルバインディングとクラスターバインディングについての違いに留意してください。ローカルのロールバインディングを使用して **cluster-admin** ロールをユーザーにバインドする場合、このユーザーがクラスター管理者の特権を持っているように表示されますが、実際にはそうではありません。一方、特定プロジェクトにバインドされる cluster-admin クラスターロールはそのプロジェクトのスーパー管理者のような機能があり、クラスターロール admin のパーミッションを付与するほか、レート制限を編集する機能などのいくつかの追加パーミッションを付与します。一方、**cluster-admin** をプロジェクトのユーザーにバインドすると、そのプロジェクトにのみ有効なスーパー管理者の権限がそのユーザーに付与されます。そのユーザーはクラスターロール **admin** のパーミッションを有するほか、レート制限を編集する機能などの、そのプロジェクトについてのいくつかの追加パーミッションを持ちます。このバインディングは、クラスター管理者にバインドされるクラスターのロールバインディングを一覧表示しない Web コンソール UI を使うと分かりにくくなります。ただし、これは、**cluster-admin** をローカルにバインドするために使用するローカルのロールバインディングを一覧表示します。

クラスターロール、クラスターロールのバインディング、ローカルロールのバインディング、ユーザー、グループおよびサービスアカウントの関係は以下に説明されています。



### 5.2.1.2. 認可の評価

OpenShift Container Platform は以下を使って認可を評価します。

#### アイデンティティー

ユーザーが属するユーザー名とグループの一覧。

#### アクション

実行する動作。ほとんどの場合、これは以下で設定されます。

- **プロジェクト:** アクセスするプロジェクト。プロジェクトは追加のアノテーションを含む Kubernetes namespace であり、これにより、ユーザーのコミュニティは、他のコミュニティと分離された状態で独自のコンテンツを編成し、管理できます。
- **動詞:** **get**、**list**、**create**、**update**、**delete**、**deletecollection**、または **watch** などのアクション自体。
- **リソース名:** アクセスする API エンドポイント。

#### バインディング

バインディングの詳細な一覧、ロールを持つユーザーまたはグループ間の関連付け。

OpenShift Container Platform は以下の手順を使って認可を評価します。

1. アイデンティティーおよびプロジェクトでスコープ設定されたアクションは、ユーザーおよびそれらのグループに適用されるすべてのバインディングを検索します。
2. バインディングは、適用されるすべてのロールを見つけるために使用されます。

3. ロールは、適用されるすべてのルールを見つけるために使用されます。
4. 一致を見つけるために、アクションが各ルールに対してチェックされます。
5. 一致するルールが見つからない場合、アクションはデフォルトで拒否されます。

## ヒント

ユーザーおよびグループは一度に複数のロールに関連付けたり、バインドしたりできることに留意してください。

プロジェクト管理者は CLI を使用してローカルロールとローカルバインディングを表示できます。これには、それぞれのロールが関連付けられる動詞およびリソースのマトリクスが含まれます。



### 重要

プロジェクト管理者にバインドされるクラスターロールは、ローカルバインディングによってプロジェクト内で制限されます。これは、**cluster-admin** または **system:admin** に付与されるクラスターロールのようにクラスター全体でバインドされる訳ではありません。

クラスターロールは、クラスターレベルで定義されるロールですが、クラスターレベルまたはプロジェクトレベルのいずれかでバインドできます。

#### 5.2.1.2.1. クラスターロールの集計

デフォルトのクラスターの管理、編集および cluster-reader ロールは、[クラスターロールの集計](#) をサポートします。ここでは、各ロールのクラスタールールは新規ルートの作成時に動的に更新されます。この機能は、カスタムリソースを作成して Kubernetes API を拡張する場合にのみ適用できます。

#### 5.2.2. プロジェクトおよび namespace

Kubernetes **namespace** は、クラスターでスコープ設定するメカニズムを提供します。namespace の詳細は、[Kubernetes ドキュメント](#) を参照してください。

Namespace は以下の一意のスコープを提供します。

- 基本的な命名の衝突を避けるための名前付きリソース。
- 信頼されるユーザーに委任された管理権限。
- コミュニティリソースの消費を制限する機能。

システム内の大半のオブジェクトのスコープは namespace で設定されますが、一部はノードやユーザーを含め、除外され、namespace が設定されません。

**プロジェクト** は追加のアノテーションを持つ Kubernetes namespace であり、通常ユーザーのリソースへのアクセスが管理される中心的な手段です。プロジェクトはユーザーのコミュニティが他のコミュニティとは切り離してコンテンツを編成し、管理することを許可します。ユーザーには、管理者によってプロジェクトへのアクセスが付与される必要があり、許可される場合はプロジェクトを作成でき、それらの独自のプロジェクトへのアクセスが自動的に付与されます。

プロジェクトには、別個の **name**、**displayName**、および **description** を設定できます。

- 必須の **name** はプロジェクトの一意的 ID であり、CLI ツールまたは API を使用する場合に最も明確に表示されます。名前の最大長さは 63 文字です。
- オプションの **displayName** は、Web コンソールでのプロジェクトの表示方法を示します (デフォルトは **name** に設定される)。
- オプションの **description** には、プロジェクトのさらに詳細な記述を使用でき、これも Web コンソールで表示できます。

各プロジェクトは、以下の独自のセットのスコープを設定します。

オブジェクト	説明
<b>Objects</b>	Pod、サービス、レプリケーションコントローラーなど。
<b>Policies</b>	ユーザーがオブジェクトに対してアクションを実行できるか、できないかについてのルール。
<b>Constraints</b>	制限を設定できるそれぞれの種類のオブジェクトのクォータ。
<b>service account (サービスアカウント)</b>	サービスアカウントは、プロジェクトのオブジェクトへの指定されたアクセスで自動的に機能します。

クラスター管理者はプロジェクトを作成でき、プロジェクトの管理者権限をユーザーコミュニティの任意のメンバーに委任できます。クラスター管理者は、開発者が独自のプロジェクトを作成することも許可できます。

開発者および管理者は、CLI または Web コンソールを使用してプロジェクトとの対話を実行できます。

### 5.2.3. デフォルトプロジェクト

OpenShift Container Platform にはデフォルトのプロジェクトが多数含まれ、**openshift-** で始まるプロジェクトはユーザーにとって最も重要になります。これらのプロジェクトは、Pod として実行されるマスターコンポーネントおよび他のインフラストラクチャーコンポーネントをホストします。[Critical Pod アノテーション](#)を持つこれらの namespace で作成される Pod は Critical (重要) とみなされ、kubelet による受付が保証されます。これらの namespace のマスターコンポーネント用に作成された Pod には、すでに Critical のマークが付けられています。



#### 注記

デフォルト namespace (**default**、**kube-system**、**kube-public**、**openshift-node**、**openshift-infra**、**openshift**) のいずれかに作成された Pod に SCC を割り当てることはできません。これらの namespace は Pod またはサービスを実行するために使用することはできません。

### 5.2.4. クラスターロールおよびバインディングの表示

**oc** CLI で **oc describe** コマンドを使用して、クラスターロールおよびバインディングを表示できます。

前提条件

- **oc** CLI をインストールします。
- クラスターロールおよびバインディングを表示するパーミッションを取得します。

クラスター全体でバインドされた **cluster-admin** のデフォルトのクラスターロールを持つユーザーは、クラスターロールおよびバインディングの表示を含む、すべてのリソースでのすべてのアクションを実行できます。

手順

1. クラスターロールおよびそれらの関連付けられたルールセットを表示するには、以下を実行します。

```
$ oc describe clusterrole.rbac
```

出力例

```
Name:      admin
Labels:    kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                Non-Resource URLs  Resource Names  Verbs
-----
.packages.apps.redhat.com                  []                 []              [* create update
patch delete get list watch]
imagestreams                               []                 []              [create delete
deletecollection get list patch update watch create get list watch]
imagestreams.image.openshift.io            []                 []              [create delete
deletecollection get list patch update watch create get list watch]
secrets                                    []                 []              [create delete deletecollection
get list patch update watch get list watch create delete deletecollection patch update]
buildconfigs/webhooks                      []                 []              [create delete
deletecollection get list patch update watch get list watch]
buildconfigs                              []                 []              [create delete
deletecollection get list patch update watch get list watch]
buildlogs                                  []                 []              [create delete deletecollection
get list patch update watch get list watch]
deploymentconfigs/scale                    []                 []              [create delete
deletecollection get list patch update watch get list watch]
deploymentconfigs                         []                 []              [create delete
deletecollection get list patch update watch get list watch]
imagestreamimages                         []                 []              [create delete
deletecollection get list patch update watch get list watch]
imagestreammappings                       []                 []              [create delete
deletecollection get list patch update watch get list watch]
imagestreamtags                           []                 []              [create delete
deletecollection get list patch update watch get list watch]
processedtemplates                        []                 []              [create delete
deletecollection get list patch update watch get list watch]
routes                                    []                 []              [create delete deletecollection
get list patch update watch get list watch]
templateconfigs                           []                 []              [create delete
deletecollection get list patch update watch get list watch]
```

templateinstances	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
templates	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
deploymentconfigs.apps.openshift.io/scale	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
deploymentconfigs.apps.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
buildconfigs.build.openshift.io/webhooks	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
buildconfigs.build.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
buildlogs.build.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
imagestreamimages.image.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
imagestreammappings.image.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
imagestreamtags.image.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
routes.route.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
processedtemplates.template.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
templateconfigs.template.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
templateinstances.template.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
templates.template.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch get list watch]			
serviceaccounts	[]	[]	[create delete]
deletecollection get list patch update watch impersonate create delete deletecollection patch update get list watch]			
imagestreams/secrets	[]	[]	[create delete]
deletecollection get list patch update watch]			
rolebindings	[]	[]	[create delete]
deletecollection get list patch update watch]			
roles	[]	[]	[create delete deletecollection get list patch update watch]
rolebindings.authorization.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch]			
roles.authorization.openshift.io	[]	[]	[create delete]
deletecollection get list patch update watch]			
imagestreams.image.openshift.io/secrets	[]	[]	[create delete]
deletecollection get list patch update watch]			
rolebindings.rbac.authorization.k8s.io	[]	[]	[create delete]
deletecollection get list patch update watch]			
roles.rbac.authorization.k8s.io	[]	[]	[create delete]
deletecollection get list patch update watch]			
networkpolicies.extensions	[]	[]	[create delete]
deletecollection patch update create delete deletecollection get list patch update watch get list watch]			
networkpolicies.networking.k8s.io	[]	[]	[create delete]
deletecollection patch update create delete deletecollection get list patch update watch get list watch]			
configmaps	[]	[]	[create delete]



deletecollection patch update get list watch]			
endpoints	[]	[]	[create delete
deletecollection patch update get list watch]			
persistentvolumeclaims	[]	[]	[create delete
deletecollection patch update get list watch]			
Pods	[]	[]	[create delete deletecollection
patch update get list watch]			
replicationcontrollers/scale	[]	[]	[create delete
deletecollection patch update get list watch]			
replicationcontrollers	[]	[]	[create delete
deletecollection patch update get list watch]			
services	[]	[]	[create delete deletecollection
patch update get list watch]			
daemonsets.apps	[]	[]	[create delete
deletecollection patch update get list watch]			
deployments.apps/scale	[]	[]	[create delete
deletecollection patch update get list watch]			
deployments.apps	[]	[]	[create delete
deletecollection patch update get list watch]			
replicasets.apps/scale	[]	[]	[create delete
deletecollection patch update get list watch]			
replicasets.apps	[]	[]	[create delete
deletecollection patch update get list watch]			
statefulsets.apps/scale	[]	[]	[create delete
deletecollection patch update get list watch]			
statefulsets.apps	[]	[]	[create delete
deletecollection patch update get list watch]			
horizontalpodautoscalers.autoscaling		[]	[create delete
deletecollection patch update get list watch]			
cronjobs.batch	[]	[]	[create delete
deletecollection patch update get list watch]			
jobs.batch	[]	[]	[create delete
deletecollection patch update get list watch]			
daemonsets.extensions	[]	[]	[create delete
deletecollection patch update get list watch]			
deployments.extensions/scale		[]	[create delete
deletecollection patch update get list watch]			
deployments.extensions	[]	[]	[create delete
deletecollection patch update get list watch]			
ingresses.extensions	[]	[]	[create delete
deletecollection patch update get list watch]			
replicasets.extensions/scale		[]	[create delete
deletecollection patch update get list watch]			
replicasets.extensions	[]	[]	[create delete
deletecollection patch update get list watch]			
replicationcontrollers.extensions/scale		[]	[create delete
deletecollection patch update get list watch]			
poddisruptionbudgets.policy	[]	[]	[create delete
deletecollection patch update get list watch]			
deployments.apps/rollback	[]	[]	[create delete
deletecollection patch update]			
deployments.extensions/rollback		[]	[create delete
deletecollection patch update]			
catalogsources.operators.coreos.com		[]	[create update
patch delete get list watch]			
clusterserviceversions.operators.coreos.com		[]	[create update

patch delete get list watch]			
installplans.operators.coreos.com	[]	[]	[create update
patch delete get list watch]			
packagemanifests.operators.coreos.com	[]	[]	[create update
patch delete get list watch]			
subscriptions.operators.coreos.com	[]	[]	[create update
patch delete get list watch]			
buildconfigs/instantiate	[]	[]	[create]
buildconfigs/instantiatebinary	[]	[]	[create]
builds/clone	[]	[]	[create]
deploymentconfigrollbacks	[]	[]	[create]
deploymentconfigs/instantiate	[]	[]	[create]
deploymentconfigs/rollback	[]	[]	[create]
imagestreamimports	[]	[]	[create]
localresourceaccessreviews	[]	[]	[create]
localsubjectaccessreviews	[]	[]	[create]
podsecuritypolicyreviews	[]	[]	[create]
podsecuritypolicyselfsubjectreviews	[]	[]	[create]
podsecuritypolicysubjectreviews	[]	[]	[create]
resourceaccessreviews	[]	[]	[create]
routes/custom-host	[]	[]	[create]
subjectaccessreviews	[]	[]	[create]
subjectrulesreviews	[]	[]	[create]
deploymentconfigrollbacks.apps.openshift.io	[]	[]	[create]
deploymentconfigs.apps.openshift.io/instantiate	[]	[]	[create]
deploymentconfigs.apps.openshift.io/rollback	[]	[]	[create]
localsubjectaccessreviews.authorization.k8s.io	[]	[]	[create]
localresourceaccessreviews.authorization.openshift.io	[]	[]	[create]
localsubjectaccessreviews.authorization.openshift.io	[]	[]	[create]
resourceaccessreviews.authorization.openshift.io	[]	[]	[create]
subjectaccessreviews.authorization.openshift.io	[]	[]	[create]
subjectrulesreviews.authorization.openshift.io	[]	[]	[create]
buildconfigs.build.openshift.io/instantiate	[]	[]	[create]
buildconfigs.build.openshift.io/instantiatebinary	[]	[]	[create]
builds.build.openshift.io/clone	[]	[]	[create]
imagestreamimports.image.openshift.io	[]	[]	[create]
routes.route.openshift.io/custom-host	[]	[]	[create]
podsecuritypolicyreviews.security.openshift.io	[]	[]	[create]
podsecuritypolicyselfsubjectreviews.security.openshift.io	[]	[]	[create]
podsecuritypolicysubjectreviews.security.openshift.io	[]	[]	[create]
jenkins.build.openshift.io	[]	[]	[edit view view admin
edit view]			
builds	[]	[]	[get create delete
deletecollection get list patch update watch get list watch]			
builds.build.openshift.io	[]	[]	[get create delete
deletecollection get list patch update watch get list watch]			
projects	[]	[]	[get delete get delete get patch
update]			
projects.project.openshift.io	[]	[]	[get delete get delete
get patch update]			
namespaces	[]	[]	[get get list watch]
Pods/attach	[]	[]	[get list watch create delete
deletecollection patch update]			
Pods/exec	[]	[]	[get list watch create delete
deletecollection patch update]			
Pods/portforward	[]	[]	[get list watch create

delete deletecollection patch update]			
pods/proxy	[]	[]	[get list watch create delete]
deletecollection patch update]			
services/proxy	[]	[]	[get list watch create delete]
deletecollection patch update]			
routes/status	[]	[]	[get list watch update]
routes.route.openshift.io/status		[]	[get list watch update]
appliedclusterresourcequotas		[]	[get list watch]
bindings	[]	[]	[get list watch]
builds/log	[]	[]	[get list watch]
deploymentconfigs/log		[]	[get list watch]
deploymentconfigs/status		[]	[get list watch]
events	[]	[]	[get list watch]
imagestreams/status		[]	[get list watch]
limitranges	[]	[]	[get list watch]
namespaces/status		[]	[get list watch]
pods/log	[]	[]	[get list watch]
pods/status	[]	[]	[get list watch]
replicationcontrollers/status		[]	[get list watch]
resourcequotas/status		[]	[get list watch]
resourcequotas	[]	[]	[get list watch]
resourcequotausages		[]	[get list watch]
rolebindingrestrictions	[]	[]	[get list watch]
deploymentconfigs.apps.openshift.io/log		[]	[get list watch]
deploymentconfigs.apps.openshift.io/status		[]	[get list watch]
controllerrevisions.apps	[]	[]	[get list watch]
rolebindingrestrictions.authorization.openshift.io		[]	[get list watch]
builds.build.openshift.io/log	[]	[]	[get list watch]
imagestreams.image.openshift.io/status		[]	[get list watch]
appliedclusterresourcequotas.quota.openshift.io		[]	[get list watch]
imagestreams/layers	[]	[]	[get update get]
imagestreams.image.openshift.io/layers		[]	[get update get]
builds/details	[]	[]	[update]
builds.build.openshift.io/details		[]	[update]

Name: basic-user

Labels: <none>

Annotations: openshift.io/description: A user that can get basic information about projects.  
rbac.authorization.kubernetes.io/autoupdate: true

PolicyRule:

Resources	Non-Resource URLs	Resource Names	Verbs
-----	-----	-----	----
selfsubjectrulesreviews	[]	[]	[create]
selfsubjectaccessreviews.authorization.k8s.io		[]	[create]
selfsubjectrulesreviews.authorization.openshift.io	[]	[]	[create]
clusterroles.rbac.authorization.k8s.io	[]	[]	[get list watch]
clusterroles	[]	[]	[get list]
clusterroles.authorization.openshift.io	[]	[]	[get list]
storageclasses.storage.k8s.io	[]	[]	[get list]
users	[]	[~]	[get]
users.user.openshift.io	[]	[~]	[get]
projects	[]	[]	[list watch]
projects.project.openshift.io	[]	[]	[list watch]
projectrequests	[]	[]	[list]
projectrequests.project.openshift.io	[]	[]	[list]

```

Name:      cluster-admin
Labels:    kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
Resources  Non-Resource URLs  Resource Names  Verbs
-----
*. *      []      []      [*]
          [*]      []      [*]
...

```

2. 各種のロールにバインドされたユーザーおよびグループを示す、クラスターのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe clusterrolebinding.rbac
```

## 出力例

```

Name:      alertmanager-main
Labels:    <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: alertmanager-main
Subjects:
  Kind      Name      Namespace
  ---      ---      ---
  ServiceAccount alertmanager-main openshift-monitoring

Name:      basic-users
Labels:    <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind Name      Namespace
  --- ---      ---
  Group system:authenticated

Name:      cloud-credential-operator-rolebinding
Labels:    <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: cloud-credential-operator-role
Subjects:
  Kind      Name      Namespace
  ---      ---      ---
  ServiceAccount default openshift-cloud-credential-operator

```

```

Name:      cluster-admin
Labels:    kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind  Name      Namespace
  ----  ---      -
  Group system:masters

Name:      cluster-admins
Labels:    <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind  Name      Namespace
  ----  ---      -
  Group system:cluster-admins
  User  system:admin

Name:      cluster-api-manager-rolebinding
Labels:    <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: cluster-api-manager-role
Subjects:
  Kind      Name      Namespace
  ----      ---      -
  ServiceAccount default openshift-machine-api

...

```

### 5.2.5. ローカルのロールバインディングの表示

**oc** CLI で **oc describe** コマンドを使用して、ローカルロールおよびバインディングを表示できます。

#### 前提条件

- **oc** CLI をインストールします。
- ローカルロールおよびバインディングを表示するパーミッションを取得します。
  - クラスター全体でバインドされた **cluster-admin** のデフォルトのクラスターロールを持つユーザーは、ローカルロールおよびバインディングの表示を含む、すべてのリソースでのすべてのアクションを実行できます。
  - ローカルにバインドされた **admin** のデフォルトのクラスターロールを持つユーザーは、そのプロジェクトのロールおよびバインディングを表示し、管理できます。

## 手順

1. 現在のプロジェクトの各種のロールにバインドされたユーザーおよびグループを示す、ローカルのロールバインディングの現在のセットを表示するには、以下を実行します。

```
$ oc describe rolebinding.rbac
```

2. 別のプロジェクトのローカルロールバインディングを表示するには、**-n** フラグをコマンドに追加します。

```
$ oc describe rolebinding.rbac -n joe-project
```

## 出力例

```
Name:      admin
Labels:    <none>
Annotations: <none>
```

Role:

```
Kind: ClusterRole
```

```
Name: admin
```

Subjects:

```
Kind Name      Namespace
```

```
----
```

```
User kube:admin
```

```
Name:      system:deployers
```

```
Labels:    <none>
```

```
Annotations: openshift.io/description:
```

```
Allows deploymentconfigs in this namespace to rollout pods in
this namespace. It is auto-managed by a controller; remove
subjects to disa...
```

Role:

```
Kind: ClusterRole
```

```
Name: system:deployer
```

Subjects:

```
Kind      Name      Namespace
```

```
----
```

```
ServiceAccount deployer joe-project
```

```
Name:      system:image-builders
```

```
Labels:    <none>
```

```
Annotations: openshift.io/description:
```

```
Allows builds in this namespace to push images to this
namespace. It is auto-managed by a controller; remove subjects
to disable.
```

Role:

```
Kind: ClusterRole
```

```
Name: system:image-builder
```

Subjects:

```
Kind      Name      Namespace
```

```
----
```

```
ServiceAccount builder joe-project
```

```

Name:      system:image-pullers
Labels:    <none>
Annotations: openshift.io/description:
            Allows all pods in this namespace to pull images from this
            namespace. It is auto-managed by a controller; remove subjects
            to disable.
Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind  Name                      Namespace
  ----  ---                      -
  Group system:serviceaccounts:joe-project

```

### 5.2.6. ロールのユーザーへの追加

**oc adm** 管理者 CLI を使用してロールおよびバインディングを管理できます。

ロールをユーザーまたはグループにバインドするか、または追加することにより、そのロールによって付与されるアクセスがそのユーザーまたはグループに付与されます。**oc adm policy** コマンドを使用して、ロールのユーザーおよびグループへの追加、またはユーザーおよびグループからの削除を行うことができます。

デフォルトのクラスターロールのすべてを、プロジェクト内のローカルユーザーまたはグループにバインドできます。

#### 手順

1. ロールを特定プロジェクトのユーザーに追加します。

```
$ oc adm policy add-role-to-user <role> <user> -n <project>
```

たとえば、以下を実行して **admin** ロールを **joe** プロジェクトの **alice** ユーザーに追加できます。

```
$ oc adm policy add-role-to-user admin alice -n joe
```

2. 出力でローカルロールバインディングを確認し、追加の内容を確認します。

```
$ oc describe rolebinding.rbac -n <project>
```

たとえば、**joe** プロジェクトのローカルロールバインディングを表示するには、以下を実行します。

```
$ oc describe rolebinding.rbac -n joe
```

#### 出力例

```

Name:      admin
Labels:    <none>
Annotations: <none>
Role:

```

```

Kind: ClusterRole
Name: admin
Subjects:
  Kind Name      Namespace
  ---- ----      -
  User kube:admin

```

```

Name:      admin-0
Labels:    <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:
  Kind Name      Namespace
  ---- ----      -
  User alice 1

```

```

Name:      system:deployers
Labels:    <none>
Annotations: openshift.io/description:
            Allows deploymentconfigs in this namespace to rollout pods in
            this namespace. It is auto-managed by a controller; remove
            subjects to disa...
Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind      Name      Namespace
  ----      -
  ServiceAccount deployer joe

```

```

Name:      system:image-builders
Labels:    <none>
Annotations: openshift.io/description:
            Allows builds in this namespace to push images to this
            namespace. It is auto-managed by a controller; remove subjects
            to disable.
Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind      Name      Namespace
  ----      -
  ServiceAccount builder joe

```

```

Name:      system:image-pullers
Labels:    <none>
Annotations: openshift.io/description:
            Allows all pods in this namespace to pull images from this
            namespace. It is auto-managed by a controller; remove subjects
            to disable.

```



```

Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind  Name                      Namespace
  ----  ---                      -
  Group system:serviceaccounts:joe

```

- 1 **alice** ユーザーが **admins RoleBinding** に追加されています。

### 5.2.7. ローカルロールの作成

プロジェクトのローカルロールを作成し、これをユーザーにバインドできます。

#### 手順

1. プロジェクトのローカルロールを作成するには、以下のコマンドを実行します。

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
```

このコマンドで以下を指定します。

- **<name>**: ローカルのロール名です。
- **<verb>**: ロールに適用する動詞のコンマ区切りの一覧です。
- **<resource>**: ロールが適用されるリソースです。
- **<project>** (プロジェクト名)

たとえば、ユーザーが **blue** プロジェクトで Pod を閲覧できるようにするローカルロールを作成するには、以下のコマンドを実行します。

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. 新規ロールをユーザーにバインドするには、以下のコマンドを実行します。

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
```

### 5.2.8. クラスターロールの作成

クラスターロールを作成できます。

#### 手順

1. クラスターロールを作成するには、以下のコマンドを実行します。

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

このコマンドで以下を指定します。

- **<name>**: ローカルのロール名です。

- **<verb>**: ロールに適用する動詞のコンマ区切りの一覧です。
- **<resource>**: ロールが適用されるリソースです。

たとえば、ユーザーが Pod を閲覧できるようにするクラスターロールを作成するには、以下のコマンドを実行します。

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

### 5.2.9. ローカルロールバインディングのコマンド

以下の操作を使用し、ローカルのロールバインディングでのユーザーまたはグループの関連付けられたロールを管理する際に、プロジェクトは **-n** フラグで指定できます。これが指定されていない場合には、現在のプロジェクトが使用されます。

ローカル RBAC 管理に以下のコマンドを使用できます。

表5.1 ローカルのロールバインディング操作

コマンド	説明
<b>\$ oc adm policy who-can &lt;verb&gt; &lt;resource&gt;</b>	リソースに対してアクションを実行できるユーザーを示します。
<b>\$ oc adm policy add-role-to-user &lt;role&gt; &lt;username&gt;</b>	指定されたロールを現在のプロジェクトの指定ユーザーにバインドします。
<b>\$ oc adm policy remove-role-from-user &lt;role&gt; &lt;username&gt;</b>	現在のプロジェクトの指定ユーザーから指定されたロールを削除します。
<b>\$ oc adm policy remove-user &lt;username&gt;</b>	現在のプロジェクトの指定ユーザーとそれらのロールのすべてを削除します。
<b>\$ oc adm policy add-role-to-group &lt;role&gt; &lt;groupname&gt;</b>	指定されたロールを現在のプロジェクトの指定グループにバインドします。
<b>\$ oc adm policy remove-role-from-group &lt;role&gt; &lt;groupname&gt;</b>	現在のプロジェクトの指定グループから指定されたロールを削除します。
<b>\$ oc adm policy remove-group &lt;groupname&gt;</b>	現在のプロジェクトの指定グループとそれらのロールのすべてを削除します。

### 5.2.10. クラスターのロールバインディングコマンド

以下の操作を使用して、クラスターのロールバインディングも管理できます。クラスターのロールバインディングは namespace を使用していないリソースを使用するため、**-n** フラグはこれらの操作に使用されません。

表5.2 クラスターのロールバインディング操作

コマンド	説明
<code>\$ oc adm policy add-cluster-role-to-user &lt;role&gt; &lt;username&gt;</code>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーにバインドします。
<code>\$ oc adm policy remove-cluster-role-from-user &lt;role&gt; &lt;username&gt;</code>	指定されたロールをクラスターのすべてのプロジェクトの指定ユーザーから削除します。
<code>\$ oc adm policy add-cluster-role-to-group &lt;role&gt; &lt;groupname&gt;</code>	指定されたロールをクラスターのすべてのプロジェクトの指定グループにバインドします。
<code>\$ oc adm policy remove-cluster-role-from-group &lt;role&gt; &lt;groupname&gt;</code>	指定されたロールをクラスターのすべてのプロジェクトの指定グループから削除します。

### 5.2.11. クラスター管理者の作成

**cluster-admin** ロールは、クラスターリソースの変更など、OpenShift Container Platform クラスターでの管理者レベルのタスクを実行するために必要です。

#### 前提条件

- クラスター管理者として定義するユーザーを作成していること。

#### 手順

- ユーザーをクラスター管理者として定義します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user>
```

## 5.3. KUBEADMIN ユーザー

OpenShift Container Platform は、インストールプロセスの完了後にクラスター管理者 **kubeadmin** を作成します。

このユーザーには、**cluster-admin** ロールが自動的に適用され、このユーザーはクラスターの root ユーザーとしてみなされます。パスワードは動的に生成され、OpenShift Container Platform 環境に対して一意です。インストールの完了後に、パスワードはインストールプログラムの出力で提供されます。以下に例を示します。

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster
with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

### 5.3.1. kubeadmin ユーザーの削除

アイデンティティープロバイダーを定義し、新規 **cluster-admin** ユーザーを作成した後に、クラスターのセキュリティを強化するために **kubeadmin** を削除できます。

**警告**

別のユーザーが **cluster-admin** になる前にこの手順を実行する場合、OpenShift Container Platform は再インストールされる必要があります。このコマンドをやり直すことはできません。

**前提条件**

- 1つ以上のアイデンティティプロバイダーを設定しておく必要があります。
- **cluster-admin** ロールをユーザーに追加しておく必要があります。
- 管理者としてログインしている必要があります。

**手順**

- **kubeadmin** シークレットを削除します。

```
$ oc delete secrets kubeadmin -n kube-system
```

## 5.4. イメージ設定リソース

イメージレジストリーの設定について理解し、これを設定します。

### 5.4.1. イメージコントローラー設定パラメーター

**image.config.openshift.io/cluster** resource は、イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。**spec** は以下の設定パラメーターを提供します。

パラメーター	説明
<b>allowedRegistriesForImport</b>	<p>標準ユーザーがイメージのインポートに使用できるコンテナイメージレジストリーを制限します。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは <b>ImageStreamMappings</b> を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。</p> <p>このリストのすべての要素に、レジストリーのドメイン名で指定されるレジストリーの場所が含まれます。</p> <p><b>domainName:</b> レジストリーのドメイン名を指定します。レジストリーが標準以外の (<b>80</b> または <b>443</b>) ポートを使用する場合、ポートはドメイン名にも含まれる必要があります。</p> <p><b>insecure:</b> insecure はレジストリーがセキュアか、または非セキュアであることを示します。指定がない場合には、デフォルトでレジストリーはセキュアであることが想定されます。</p>

パラメーター	説明
<b>additionalTrustedCA</b>	<p><b>image stream import</b>、<b>pod image pull</b>、<b>openshift-image-registry pullthrough</b>、およびビルド時に信頼される必要のある追加の CA が含まれる設定マップの参照です。</p> <p>この設定マップの namespace は <b>openshift-config</b> です。設定マップの形式では、信頼する追加のレジストリー CA についてレジストリーのホスト名をキーとして使用し、PEM エンコード証明書を値として使用します。</p>
<b>externalRegistryHostnames</b>	<p>デフォルトの外部イメージレジストリーのホスト名を指定します。外部ホスト名は、イメージレジストリーが外部に公開される場合にのみ設定される必要があります。最初の値は、イメージストリームの <b>publicDockerImageRepository</b> フィールドで使用されます。値は <b>hostname[:port]</b> 形式の値である必要があります。</p>
<b>registrySources</b>	<p>コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。たとえば、非セキュアなアクセスを許可するかどうかを設定します。内部クラスターレジストリーの設定は含まれません。</p> <p><b>insecureRegistries</b>: 有効な TLS 証明書を持たないか、または HTTP 接続のみをサポートするレジストリーです。</p> <p><b>blockedRegistries</b>: イメージのプルおよびプッシュアクションについて拒否リストに指定します。他のすべてのレジストリーは許可されます。</p> <p><b>allowedRegistries</b>: イメージのプルおよびプッシュアクションについて許可リストに指定します。他のすべてのレジストリーはブロックされます。</p> <p><b>blockedRegistries</b> または <b>allowedRegistries</b> のいずれかを設定できますが、両方を設定することはできません。</p>



### 警告

**allowedRegistries** パラメーターが定義されると、明示的に一覧表示されない限り、**registry.redhat.io** レジストリーと **quay.io** レジストリー、およびデフォルトの内部イメージレジストリーを含むすべてのレジストリーがブロックされます。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を含むすべてのレジストリーを **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージで必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

**image.config.openshift.io/cluster** リソースの **status** フィールドは、クラスターから観察される値を保持します。

パラメーター	説明
<b>internalRegistryHostname</b>	<b>internalRegistryHostname</b> を制御するイメージレジストリー Operator によって設定されます。これはデフォルトの内部イメージレジストリーのホスト名を設定します。値は <b>hostname[:port]</b> 形式の値である必要があります。後方互換性を確保するために、 <b>OPENSHIFT_DEFAULT_REGISTRY</b> 環境変数を依然として使用できますが、この設定によってこの環境変数は上書きされます。
<b>externalRegistryHostnames</b>	イメージレジストリー Operator によって設定され、外部に公開される際にイメージレジストリーの外部のホスト名を指定します。最初の値は、イメージストリームの <b>publicDockerImageRepository</b> フィールドで使用されます。値は <b>hostname[:port]</b> 形式の値である必要があります。

### 5.4.2. イメージ設定内容の設定

**image.config.openshift.io/cluster** カスタムリソース (CR) を編集してイメージレジストリーの設定を行うことができます。Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。

#### 手順

1. **image.config.openshift.io/cluster** カスタムリソースを編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、**image.config.openshift.io/cluster** CR の例になります。

```
apiVersion: config.openshift.io/v1
kind: Image ❶
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: ❷
  - domainName: quay.io
    insecure: false
  additionalTrustedCA: ❸
  name: myconfigmap
  registrySources: ❹
  allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
  insecureRegistries:
```

```
- insecure.com
status:
internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- 1 **Image:** イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。
- 2 **allowedRegistriesForImport:** 標準ユーザーがイメージのインポートに使用するコンテナイメージレジストリーを制限します。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは **ImageStreamMappings** を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。
- 3 **additionalTrustedCA:** イメージストリームのインポート、Pod のイメージプル、**openshift-image-registry** プルスルー、およびビルド時に信頼される追加の認証局 (CA) が含まれる設定マップの参照です。この設定マップの namespace は **openshift-config** です。設定マップの形式では、信頼する追加のレジストリー CA についてレジストリーのホスト名をキーとして使用し、PEM 証明書を値として使用します。
- 4 **registrySources:** コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。たとえば、非セキュアなアクセスを許可するかどうかを設定します。内部クラスターレジストリーの設定は含まれません。この例では、使用可能なレジストリーを定義する **allowedRegistries** を一覧表示します。一覧表示されたレジストリーのいずれかが非セキュアです。

2. 変更が適用されたことを確認するには、ノードを一覧表示します。

```
$ oc get nodes
```

## 出力例

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-j5cd0qt-f76d1-vfj5x-master-0 v1.19.0+7070803	Ready		master	98m
ci-ln-j5cd0qt-f76d1-vfj5x-master-1 v1.19.0+7070803	Ready,SchedulingDisabled		master	99m
ci-ln-j5cd0qt-f76d1-vfj5x-master-2 v1.19.0+7070803	Ready		master	98m
ci-ln-j5cd0qt-f76d1-vfj5x-worker-b-nsnd4 v1.19.0+7070803	Ready		worker	90m
ci-ln-j5cd0qt-f76d1-vfj5x-worker-c-5z2gz v1.19.0+7070803	NotReady,SchedulingDisabled		worker	90m
ci-ln-j5cd0qt-f76d1-vfj5x-worker-d-stsjv v1.19.0+7070803	Ready		worker	90m

### 5.4.2.1. イメージレジストリーアクセス用の追加のトラストストアの設定

**image.config.openshift.io/cluster** カスタムリソースには、イメージレジストリーのアクセス時に信頼される追加の認証局が含まれる設定マップへの参照を含めることができます。

#### 前提条件

- 認証局 (CA) は PEM でエンコードされている必要があります。

## 手順

設定マップを **openshift-config** namespace に作成し、その名前を **image.config.openshift.io** カスタムリソースの **AdditionalTrustedCA** で使用し、追加の CA を指定することができます。

設定マップキーは、この CA が信頼されるポートを持つレジストリーのホスト名であり、base64 エンコード証明書が信頼する追加の各レジストリー CA についての値になります。

### イメージレジストリー CA の設定マップの例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com:5000: | ❶
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

- ❶ レジストリーにポートがある場合 (例: **registry-with-port.example.com:5000**)、: は .. に置き換える必要があります。

以下の手順で追加の CA を設定することができます。

1. 追加の CA を設定するには、以下を実行します。

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

#### 5.4.2.2. 非セキュアなレジストリー

**image.config.openshift.io/cluster** カスタムリソース (CR) を編集して、非セキュアなレジストリーを追加できます。OpenShift Container Platform は、この CR への変更をクラスター内のすべてのノードに適用します。

有効な SSL 証明書を使用しないレジストリー、または HTTPS 接続を必要としないレジストリーは、非セキュアであると見なされます。





### 警告

セキュリティ上のリスクを軽減するために、非セキュアな外部レジストリーは回避する必要があります。

## 手順

1. **image.config.openshift.io/cluster** CR を編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、非セキュアなレジストリーのリストを含む **image.config.openshift.io/cluster** CR の例になります。

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: ❶
  insecureRegistries: ❷
  - insecure.com
  allowedRegistries:
  - example.com
  - quay.io
  - registry.redhat.io
  - insecure.com ❸
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- ❶ **registrySources**: コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。内部クラスターレジストリーの設定は含まれません。
- ❷ 非セキュアなレジストリーを指定します。
- ❸ 非セキュアなレジストリーが **allowedRegistries** 一覧に含まれていることを確認します。



## 注記

**allowedRegistries** パラメーターが定義されると、明示的に一覧表示されない限り、registry.redhat.io レジストリーと quay.io レジストリー、およびデフォルトの内部イメージレジストリーを含むすべてのレジストリーがブロックされます。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を含むすべてのレジストリーを **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージで必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。非セキュアでブロックされたレジストリーへの変更は、各ノードの **/etc/containers/registries.conf** ファイルに表示されます。

2. レジストリーがポリシーファイルに追加されていることを確認するには、ノードで以下のコマンドを使用します。

```
$ cat /host/etc/containers/registries.conf
```

以下の例は、**insecure.com** レジストリーからのイメージが非セキュアであり、イメージのプルおよびプッシュで許可されることを示しています。

## 出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "insecure.com"
  insecure = true
```

### 5.4.2.3. イメージレジストリーのリポジトリミラーリングの設定

コンテナレジストリーのリポジトリミラーリングの設定により、以下が可能になります。

- ソースイメージのレジストリーのリポジトリからイメージをプルする要求をリダイレクトするように OpenShift Container Platform クラスターを設定し、これをミラーリングされたイメージレジストリーのリポジトリで解決できるようにします。
- 各ターゲットリポジトリに対して複数のミラーリングされたリポジトリを特定し、1つのミラーがダウンした場合に別のミラーを使用できるようにします。

以下は、OpenShift Container Platform のリポジトリミラーリングの属性の一部です。

- イメージプルには、レジストリーのダウンタイムに対する回復性があります。
- ネットワークが制限された環境のクラスターは、重要な場所 (quay.io など) からイメージをプルでき、会社のファイアウォールの背後にあるレジストリーが要求されたイメージを提供するようにできます。
- イメージのプル要求時にレジストリーへの接続が特定の順序で試行され、通常は永続レジストリーが最後に試行されます。

- 入力したミラー情報は、OpenShift Container Platform クラスターの全ノードの `/etc/containers/registries.conf` ファイルに追加されます。
- ノードがソースリポジトリからイメージの要求を行うと、要求されたコンテンツを見つけるまで、ミラーリングされた各リポジトリに対する接続を順番に試行します。すべてのミラーで障害が発生した場合、クラスターはソースリポジトリに対して試行します。成功すると、イメージはノードにプルされます。

リポジトリミラーリングのセットアップは次の方法で実行できます。

- OpenShift Container Platform のインストール時:  
OpenShift Container Platform が必要とするコンテナイメージをプルし、それらのイメージを会社のファイアウォールの内側に配置すると、制限されたネットワーク内にあるデータセンターに OpenShift Container Platform をインストールできます。
- OpenShift Container Platform の新規インストール後:  
OpenShift Container Platform インストール時にミラーリングを設定しなくても、**ImageContentSourcePolicy** オブジェクトを使用して後で設定することができます。

以下の手順では、インストール後のミラーを設定し、以下を識別する **ImageContentSourcePolicy** オブジェクトを作成します。

- ミラーリングするコンテナイメージリポジトリのソース
- ソースリポジトリから要求されたコンテンツを提供する各ミラーリポジトリの個別のエントリ。



#### 注記

**ImageContentSourcePolicy** オブジェクトを持つクラスターのグローバルプルシークレットのみを設定できます。プロジェクトにプルシークレットを追加することはできません。

#### 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. ミラーリングされたりポジトリを設定します。以下のいずれかを実行します。
  - [Repository Mirroring in Red Hat Quay](#) で説明されているように、Red Hat Quay でミラーリングされたりポジトリを設定します。Red Hat Quay を使用すると、あるリポジトリから別のリポジトリにイメージをコピーでき、これらのリポジトリを一定期間繰り返し自動的に同期することもできます。
  - **skopeo** などのツールを使用して、ソースディレクトリからミラーリングされたりポジトリにイメージを手動でコピーします。  
たとえば、Red Hat Enterprise Linux (RHEL 7 または RHEL 8) システムに **skopeo** RPM パッケージをインストールした後、以下の例に示すように **skopeo** コマンドを使用します。

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
```

```
a6 \
docker://example.io/example/ubi-minimal
```

この例では、**example.io** という名前のコンテナイメージレジストリーと **example** という名前のイメージリポジトリがあり、そこに **registry.access.redhat.com** から **ubi8/ubi-minimal** イメージをコピーします。レジストリーを作成した後、OpenShift Container Platform クラスターを設定して、ソースリポジトリで作成される要求をミラーリングされたりリポジトリにリダイレクトできます。

2. OpenShift Container Platform クラスターにログインします。
3. **ImageContentSourcePolicy** ファイル (例: **registryrepomirror.yaml**) を作成し、ソースとミラーを固有のレジストリー、およびリポジトリのペアとイメージのものに置き換えます。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal ❶
    source: registry.access.redhat.com/ubi8/ubi-minimal ❷
  - mirrors:
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
```

- ❶ イメージレジストリーおよびリポジトリの名前を示します。
- ❷ ミラーリングされているコンテンツが含まれるレジストリーおよびリポジトリを示します。

4. 新しい **ImageContentSourcePolicy** オブジェクトを作成します。

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

**ImageContentSourcePolicy** オブジェクトが作成されると、新しい設定が各ノードにデプロイされ、クラスターはソースリポジトリへの要求のためにミラーリングされたりリポジトリの使用を開始します。

5. ミラーリングされた設定が適用されていることを確認するには、ノードのいずれかで以下を実行します。
  - a. ノードの一覧を表示します。

```
$ oc get node
```

### 出力例

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-137-44.ec2.internal	Ready	worker	7m	v1.18.3
ip-10-0-138-148.ec2.internal	Ready	master	11m	v1.18.3
ip-10-0-139-122.ec2.internal	Ready	master	11m	v1.18.3

```
ip-10-0-147-35.ec2.internal Ready,SchedulingDisabled worker 7m v1.18.3
ip-10-0-153-12.ec2.internal Ready worker 7m v1.18.3
ip-10-0-154-10.ec2.internal Ready master 11m v1.18.3
```

変更が適用されているため、各ワーカーノードのスケジューリングが無効にされていることを確認できます。

- b. デバッグプロセスを開始し、ノードにアクセスします。

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

### 出力例

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

- c. ノードのファイルにアクセスします。

```
sh-4.2# chroot /host
```

- d. `/etc/containers/registries.conf` ファイルをチェックして、変更が行われたことを確認します。

```
sh-4.2# cat /etc/containers/registries.conf
```

### 出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
[[registry]]
  location = "registry.access.redhat.com/ubi8/"
  insecure = false
  blocked = false
  mirror-by-digest-only = true
  prefix = ""

[[registry.mirror]]
  location = "example.io/example/ubi8-minimal"
  insecure = false

[[registry.mirror]]
  location = "example.com/example/ubi8-minimal"
  insecure = false
```

- e. ソースからノードにイメージダイジェストをプルし、ミラーによって解決されているかどうかを確認します。**ImageContentSourcePolicy** オブジェクトはイメージダイジェストのみをサポートし、イメージタグはサポートしません。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6
```

## リポジトリのミラーリングのトラブルシューティング

リポジトリのミラーリング手順が説明どおりに機能しない場合は、リポジトリミラーリングの動作方法についての以下の情報を使用して、問題のトラブルシューティングを行うことができます。

- 最初に機能するミラーは、プルされるイメージを指定するために使用されます。
- メインレジストリーは、他のミラーが機能していない場合にのみ使用されます。
- システムコンテキストによって、**Insecure** フラグがフォールバックとして使用されます。
- `/etc/containers/registries.conf` ファイルの形式が最近変更されました。現在のバージョンはバージョン 2 で、TOML 形式です。

## 5.5. OPERATORHUB を使用した OPERATOR のインストール

OperatorHub は Operator を検出するためのユーザーインターフェイスです。これは Operator Lifecycle Manager (OLM) と連携し、クラスター上で Operator をインストールし、管理します。

クラスター管理者は、OpenShift Container Platform Web コンソールまたは CLI を使用して OperatorHub から Operator をインストールできます。Operator を 1 つまたは複数の namespace にサブスクライブし、Operator をクラスター上で開発者が使用できるようにできます。

インストール時に、Operator の以下の初期設定を判別する必要があります。

### インストールモード

**All namespaces on the cluster (default)** を選択して Operator をすべての namespace にインストールするか、または (利用可能な場合は) 個別の namespace を選択し、選択された namespace のみに Operator をインストールします。この例では、**All namespaces...** を選択し、Operator をすべてのユーザーおよびプロジェクトで利用可能にします。

### 更新チャンネル

Operator が複数のチャンネルで利用可能な場合、サブスクライブするチャンネルを選択できます。たとえば、(利用可能な場合に) **stable** チャンネルからデプロイするには、これを一覧から選択します。

### 承認ストラテジー

自動 (Automatic) または手動 (Manual) のいずれかの更新を選択します。

インストールされた Operator について自動更新を選択する場合、Operator の新規バージョンが選択されたチャンネルで利用可能になると、Operator Lifecycle Manager (OLM) は人の介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。

手動更新を選択する場合、Operator の新規バージョンが利用可能になると、OLM は更新要求を作成します。クラスター管理者は、Operator が新規バージョンに更新されるように更新要求を手動で承認する必要があります。

### 5.5.1. Web コンソールを使用した OperatorHub からのインストール

OpenShift Container Platform Web コンソールを使用して OperatorHub から Operator をインストールし、これをサブスクライブできます。

#### 前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。

#### 手順

1. Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. スクロールするか、またはキーワードを **Filter by keyword** ボックスに入力し、必要な Operator を見つけます。たとえば、**jaeger** と入力し、Jaeger Operator を検索します。  
また、**インフラストラクチャー機能** でオプションをフィルターすることもできます。たとえば、非接続環境 (ネットワークが制限された環境としても知られる) で機能する Operator を表示するには、**Disconnected** を選択します。
3. Operator を選択して、追加情報を表示します。



#### 注記

コミュニティー Operator を選択すると、Red Hat がコミュニティー Operator を認定していないことを警告します。続行する前に警告を確認する必要があります。

4. Operator についての情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで以下を行います。
  - a. 以下のいずれかを選択します。
    - **All namespaces on the cluster (default)**は、デフォルトの **openshift-operators** namespace で Operator をインストールし、クラスターのすべての namespace を監視し、Operator をこれらの namespace に対して利用可能にします。このオプションは常に選択可能です。
    - **A specific namespace on the cluster**では、Operator をインストールする特定の単一 namespace を選択できます。Operator は監視のみを実行し、この単一 namespace で使用されるように利用可能になります。
  - b. **Update Channel** を選択します (複数を選択できる場合)。
  - c. 前述のように、**自動 (Automatic)** または **手動 (Manual)** の承認ストラテジーを選択します。
6. **Install** をクリックし、Operator をこの OpenShift Container Platform クラスターの選択した namespace で利用可能にします。
  - a. **手動** の承認ストラテジーを選択している場合、サブスクリプションのアップグレードステータスは、そのインストール計画を確認し、承認するまで **Upgrading** のままになります。  
**Install Plan** ページでの承認後に、サブスクリプションのアップグレードステータスは **Up to date** に移行します。
  - b. **自動** の承認ストラテジーを選択している場合、アップグレードステータスは、介入なしに **Up to date** に解決するはずです。
7. サブスクリプションのアップグレードステータスが **Up to date** になった後に、**Operators → Installed Operators** を選択し、インストールされた Operator のクラスターサービスバージョン (CSV) が表示されることを確認します。その **Status** は最終的に関連する namespace で **InstallSucceeded** に解決するはずです。



## 注記

**All namespaces...** インストールモードの場合、ステータスは **openshift-operators** namespace で **InstallSucceeded** になりますが、他の namespace でチェックする場合、ステータスは **Copied** になります。

上記通りにならない場合、以下を実行します。

- a. さらにトラブルシューティングを行うために問題を報告している **Workloads → Pods** ページで、**openshift-operators** プロジェクト (または **A specific namespace...** インストールモードが選択されている場合は他の関連の namespace) の Pod のログを確認します。

## 5.5.2. CLI を使用した OperatorHub からのインストール

OpenShift Container Platform Web コンソールを使用する代わりに、CLI を使用して OperatorHub から Operator をインストールできます。**oc** コマンドを使用して、**Subscription** オブジェクトを作成または更新します。

### 前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスタにアクセスできる。
- **oc** コマンドをローカルシステムにインストールする。

### 手順

1. OperatorHub からクラスタで利用できる Operator の一覧を表示します。

```
$ oc get packagemanifests -n openshift-marketplace
```

### 出力例

```
NAME                                CATALOG           AGE
3scale-operator                    Red Hat Operators  91m
advanced-cluster-management        Red Hat Operators  91m
amq7-cert-manager                  Red Hat Operators  91m
...
couchbase-enterprise-certified     Certified Operators 91m
crunchy-postgres-operator          Certified Operators 91m
mongodb-enterprise                 Certified Operators 91m
...
etcd                               Community Operators 91m
jaeger                             Community Operators 91m
kubefed                             Community Operators 91m
...
```

必要な Operator のカタログをメモします。

2. 必要な Operator を検査して、サポートされるインストールモードおよび利用可能なチャネルを確認します。

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```



3. **OperatorGroup** で定義される Operator グループは、Operator グループと同じ namespace 内のすべての Operator に必要な RBAC アクセスを生成するターゲット namespace を選択します。

Operator をサブスクライブする namespace には、Operator のインストールモードに一致する Operator グループが必要になります (**AllNamespaces** または **SingleNamespace** モードのいずれか)。インストールする Operator が **AllNamespaces** を使用する場合、**openshift-operators** namespace には適切な Operator グループがすでに配置されます。

ただし、Operator が **SingleNamespace** モードを使用し、適切な Operator グループがない場合、それらを作成する必要があります。



#### 注記

この手順の Web コンソールバージョンでは、**SingleNamespace** モードを選択する際に、**OperatorGroup** および **Subscription** オブジェクトの作成を背後で自動的に処理します。

- a. **OperatorGroup** オブジェクト YAML ファイルを作成します (例: **operatorgroup.yaml**)。

#### OperatorGroup オブジェクトのサンプル

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace>
spec:
  targetNamespaces:
    - <namespace>
```

- b. **OperatorGroup** オブジェクトを作成します。

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

4. **Subscription** オブジェクトの YAML ファイルを作成し、namespace を Operator にサブスクライブします (例: **sub.yaml**)。

#### Subscription オブジェクトの例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators ❶
spec:
  channel: <channel_name> ❷
  name: <operator_name> ❸
  source: redhat-operators ❹
  sourceNamespace: openshift-marketplace ❺
```

- ❶ **AllNamespaces** インストールモードの使用については、**openshift-operators** namespace を指定します。それ以外の場合は、**SingleNamespace** インストールモードの使用について関連する単一の namespace を指定します。

- 2 サブスクライブするチャンネルの名前。
- 3 サブスクライブする Operator の名前。
- 4 Operator を提供するカタログソースの名前。
- 5 カタログソースの namespace。デフォルトの OperatorHub カタログソースには **openshift-marketplace** を使用します。

5. **Subscription** オブジェクトを作成します。

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

この時点で、OLM は選択した Operator を認識します。Operator のクラスターサービスバージョン (CSV) はターゲット namespace に表示され、Operator で指定される API は作成用に利用可能になります。

## 関連情報

- [About OperatorGroups](#)