



OpenShift Container Platform 4.10

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 4.10 インストール後の設定

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

法律上の通知

Copyright © 2023 Red Hat, Inc.

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® is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js® 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® 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

第1章 インストール後の設定の概要	4
1.1. インストール後に実行する設定タスク	4
第2章 プライベートクラスターの設定	7
2.1. プライベートクラスター	7
2.2. DNS をプライベートに設定する	7
2.3. INGRESS コントローラーをプライベートに設定する	9
2.4. API サーバーをプライベートに制限する	9
第3章 ベアメタルの設定	12
3.1. BARE METAL OPERATOR について	12
3.2. BAREMETALHOST リソースについて	14
3.3. BAREMETALHOST リソースの取得	25
3.4. HOSTFIRMWARESETTINGS リソースについて	27
3.5. HOSTFIRMWARESETTINGS リソースの取得	29
3.6. HOSTFIRMWARESETTINGS リソースの編集	29
3.7. HOSTFIRMWARE SETTINGS リソースが有効であることの確認	30
3.8. FIRMWARESCHEMA リソースについて	31
3.9. FIRMWARESCHEMA リソースの取得	32
第4章 インストール後のマシン設定タスク	34
4.1. MACHINE CONFIG OPERATOR について	34
4.2. MACHINECONFIG オブジェクトを使用したノードの設定	43
4.3. LINUX コントロールグループバージョン 2 (CGROUPS V2) の有効化	52
4.4. MCO 関連のカスタムリソースの設定	61
第5章 インストール後のクラスタータスク	72
5.1. 利用可能なクラスターのカスタマイズ	72
5.2. グローバルクラスターのプルシークレットの更新	74
5.3. ワーカーノードの調整	75
5.4. 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	81
5.5. 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	88
5.6. 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	90
5.7. CLUSTER AUTOSCALER について	100
5.8. MACHINE AUTOSCALER	104
5.9. FEATUREGATE の使用によるテクノロジープレビュー機能の有効化	105
5.10. ETCD タスク	110
5.11. POD の DISRUPTION BUDGET (停止状態の予算)	132
5.12. 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	135
5.13. 非接続クラスターのイメージストリームの設定	139
5.14. CLUSTER SAMPLE OPERATOR イメージストリームタグの定期的なインポートの設定	142
第6章 インストール後のノードタスク	144
6.1. RHEL コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加	144
6.2. RHCOS コンピュータマシンの OPENSIFT CONTAINER PLATFORM クラスターへの追加	151
6.3. マシンヘルスチェックのデプロイ	162
6.4. ノードホストについての推奨プラクティス	167
6.5. HUGE PAGE	179
6.6. デバイスプラグインについて	183
6.7. ティントおよび容認 (TOLERATION)	186
6.8. TOPOLOGY MANAGER	199
6.9. リソース要求とオーバーコミット	201

6.10. 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	201
6.11. ノードレベルのオーバーコミット	209
6.12. プロジェクトレベルの制限	214
6.13. 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	214
6.14. NODE TUNING OPERATOR の使用	219
6.15. ノードあたりの POD の最大数の設定	226
第7章 インストール後のネットワーク設定	229
7.1. CLUSTER NETWORK OPERATOR (CNO) の設定	229
7.2. クラスター全体のプロキシの有効化	229
7.3. DNS をプライベートに設定する	231
7.4. INGRESS クラスタートラフィックの設定	233
7.5. ノードポートサービス範囲の設定	233
7.6. ネットワークポリシーの設定	234
7.7. サポートされる構成	244
7.8. ルーティングの最適化	246
7.9. インストール後の RHOSP ネットワーク設定	248
第8章 インストール後のストレージ設定	254
8.1. 動的プロビジョニング	254
8.2. ストレージクラスの定義	255
8.3. デフォルトストレージクラスの変更	263
8.4. ストレージの最適化	264
8.5. 利用可能な永続ストレージオプション	264
8.6. 設定可能な推奨のストレージ技術	265
8.7. RED HAT OPENSIFT DATA FOUNDATION のデプロイ	267
第9章 ユーザー向けの準備	270
9.1. アイデンティティプロバイダー設定について	270
9.2. RBAC の使用によるパーミッションの定義および適用	273
9.3. KUBEADMIN ユーザー	292
9.4. イメージ設定	293
9.5. ミラーリングされた OPERATOR カタログからの OPERATORHUB の入力	303
9.6. OPERATORHUB を使用した OPERATOR のインストールについて	305
第10章 アラート通知の設定	312
10.1. 外部システムへの通知の送信	312
10.2. 関連情報	314
第11章 接続クラスターの非接続クラスターへの変換	315
11.1. ミラーレジストリーについて	315
11.2. 前提条件	316
11.3. ミラーリングのためのクラスターの準備	316
11.4. イメージのミラーリング	318
11.5. ミラーレジストリー用のクラスターの設定	320
11.6. アプリケーションが引き続き動作することの確認	323
11.7. ネットワークからクラスターを切断します。	324
11.8. パフォーマンスが低下した INSIGHTS OPERATOR の復元	324
11.9. ネットワークの復元	324
第12章 IBM Z または LINUXONE 環境での追加デバイス設定	326
12.1. MACHINE CONFIG OPERATOR (MCO) を使用した追加デバイスの設定	326
12.2. 追加のデバイスの手動設定	331
12.3. ROCE ネットワークカード	331
12.4. FCP LUN のマルチパスの有効化	332

第1章 インストール後の設定の概要

OpenShift Container Platform のインストール後に、クラスター管理者は以下のコンポーネントを設定し、カスタマイズできます。

- マシン
- ベアメタル
- Cluster
- Node
- ネットワーク
- ストレージ
- ユーザー
- アラートおよび通知

1.1. インストール後に実行する設定タスク

クラスター管理者は、以下のインストール後の設定タスクを実施できます。

- **オペレーティングシステム機能の設定:** Machine Config Operator(MCO) は **MachineConfig** オブジェクトを管理します。MCO を使用すると、OpenShift Container Platform クラスターで以下のタスクを実行できます。
  - **MachineConfig** オブジェクトの使用によるノードの設定
  - MCO 関連のカスタムリソースの設定
- **ベアメタルノードの設定:** Bare Metal Operator (BMO) は、ベアメタルホストを管理するための Kubernetes API を実装します。これは、利用可能なベアメタルホストのインベントリーを BareMetalHost カスタムリソース定義 (CRD) のインスタンスとして維持します。Bare Metal Operator は以下を行うことができます。
  - ホストのハードウェア詳細を検査し、それを対応する BareMetalHost に報告します。これには、CPU、RAM、ディスク、NIC 等に関する情報が含まれます。
  - ホストのファームウェアを検査し、BIOS 設定を行います。
  - 必要なイメージでホストをプロビジョニングします。
  - プロビジョニング前または後に、ホストのディスクコンテンツを消去します。
- **クラスター機能の設定:** クラスター管理者は、OpenShift Container Platform クラスターの主な機能の設定リソースを変更できます。これらの機能には、以下が含まれます。
  - イメージレジストリー
  - ネットワーク設定
  - イメージビルドの動作
  - アイデンティティプロバイダー

- etcd の設定
- ワークロードを処理するマシンセットの作成
- クラウドプロバイダーの認証情報の管理
- **クラスターコンポーネントのプライベートへの設定:** デフォルトでは、インストールプログラムは、一般にアクセス可能な DNS およびエンドポイントを使用して OpenShift Container Platform をプロビジョニングします。クラスターを内部ネットワーク内からのみアクセスできるようにするには、以下のコンポーネントをプライベートに設定します。
  - DNS
  - Ingress コントローラー
  - API サーバー
- **ノード操作の実施:** デフォルトでは、OpenShift Container Platform は Red Hat Enterprise Linux CoreOS (RHCOS) コンピュータマシンを使用します。クラスター管理者は、OpenShift Container Platform クラスターマシンの以下の操作を実施できます。
  - コンピュータマシンの追加および削除
  - ティントおよび容認のノードへの追加および削除
  - ノードあたりの Pod の最大数の設定
  - デバイスマネージャーの有効化
- **ネットワークの設定:** OpenShift Container Platform のインストール後、以下を設定できます。
  - Ingress クラスタトラフィック
  - ノードポートサービス範囲
  - ネットワークポリシー
  - クラスター全体のプロキシの有効化
- **ストレージの設定:** デフォルトでは、コンテナは一時ストレージまたは一時的なローカルストレージを使用して動作します。一時ストレージには有効期間の制限があります。データを長期間保存するには、永続ストレージを設定する必要があります。以下の方法のいずれかを使用してストレージを設定できます。
  - **動的プロビジョニング:** ストレージアクセスを含む異なるレベルのストレージを制御するストレージクラスを定義して作成することで、オンデマンドでストレージを動的にプロビジョニングできます。
  - **静的プロビジョニング:** Kubernetes 永続ボリュームを使用して、既存のストレージをクラスターで利用できるようにすることができます。静的プロビジョニングは、さまざまなデバイス設定とマウントオプションをサポートできます。
- **ユーザーの設定:** OAuth アクセストークンにより、ユーザーは API に対して認証を行うことができます。クラスター管理者は、次のタスクを実行するように OAuth を設定できます。
- アイデンティティプロバイダーを指定します。
- ロールベースのアクセス制御を使用して、権限を定義し、ユーザーに提供します

- Operator Hub から Operator をインストールする
- [アラートと通知の管理](#): デフォルトでは、発生するアラートは Web コンソールのアラート UI に表示されます。外部システムにアラート通知を送信するように OpenShift Container Platform を設定することもできます。

第2章 プライベートクラスターの設定

OpenShift Container Platform バージョン 4.10 クラスターのインストール後に、そのコアコンポーネントの一部を `private` に設定できます。

2.1. プライベートクラスター

デフォルトで、OpenShift Container Platform は一般にアクセス可能な DNS およびエンドポイントを使用してプロビジョニングされます。プライベートクラスターのデプロイ後に DNS、Ingress コントローラー、および API サーバーを `private` に設定できます。



重要

クラスターにパブリックサブネットがある場合、管理者により作成されたロードバランサーサービスはパブリックにアクセスできる可能性があります。クラスターのセキュリティを確保するには、これらのサービスに明示的にプライベートアノテーションが付けられていることを確認してください。

DNS

OpenShift Container Platform を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にインストールする場合、インストールプログラムは既存のパブリックゾーンにレコードを作成し、可能な場合はクラスター独自の DNS 解決用のプライベートゾーンを作成します。パブリックゾーンおよびプライベートゾーンの両方で、インストールプログラムまたはクラスターが **Ingress** オブジェクトの ***.apps**、および API サーバーの **api** の DNS エントリーを作成します。

*.apps レコードはパブリックゾーンとプライベートゾーンのどちらでも同じであるため、パブリックゾーンを削除する際に、プライベートゾーンではクラスターのすべての DNS 解決をシームレスに提供します。

Ingress コントローラー

デフォルトの **Ingress** オブジェクトはパブリックとして作成されるため、ロードバランサーはインターネットに接続され、パブリックサブネットで使用されます。デフォルト Ingress コントローラーは内部コントローラーに置き換えることができます。

API サーバー

デフォルトでは、インストールプログラムは内部トラフィックと外部トラフィックの両方で使用するための API サーバーの適切なネットワークロードバランサーを作成します。

Amazon Web Services (AWS) では、個別のパブリックロードバランサーおよびプライベートロードバランサーが作成されます。ロードバランサーは、クラスター内で使用するために追加ポートが内部で利用可能な場合を除き、常に同一です。インストールプログラムは API サーバー要件に基づいてロードバランサーを自動的に作成または破棄しますが、クラスターはそれらを管理または維持しません。クラスターの API サーバーへのアクセスを保持する限り、ロードバランサーを手動で変更または移動できます。パブリックロードバランサーの場合、ポート 6443 は開放され、ヘルスチェックが HTTPS について **/readyz** パスに対して設定されます。

Google Cloud Platform では、内部および外部 API トラフィックの両方を管理するために単一のロードバランサーが作成されるため、ロードバランサーを変更する必要はありません。

Microsoft Azure では、パブリックおよびプライベートロードバランサーの両方が作成されます。ただし、現在の実装には制限があるため、プライベートクラスターで両方のロードバランサーを保持します。

2.2. 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: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: 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: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>-wfp4: owned
status: {}

```

2.3. INGRESS コントローラーをプライベートに設定する

クラスターのデプロイ後に、その Ingress コントローラーをプライベートゾーンのみを使用するように変更できます。

手順

1. 内部エンドポイントのみを使用するようにデフォルト Ingress コントローラーを変更します。

```

$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF

```

出力例

```

ingresscontroller.operator.openshift.io "default" deleted
ingresscontroller.operator.openshift.io/default replaced

```

パブリック DNS エントリーが削除され、プライベートゾーンエントリーが更新されます。

2.4. API サーバーをプライベートに制限する

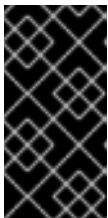
クラスターを Amazon Web Services (AWS) または Microsoft Azure にデプロイした後に、プライベートゾーンのみを使用するように API サーバーを再設定することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとして Web コンソールにアクセスできること。

手順

1. AWS または Azure の Web ポータルまたはコンソールで、以下のアクションを実行します。
  - a. 適切なロードバランサーコンポーネントを見つけ、削除します。
    - AWS の場合は、外部ロードバランサーを削除します。プライベートゾーンの API DNS エントリーは、同一の設定を使用する内部ロードバランサーをすでに参照するため、内部ロードバランサーを変更する必要はありません。
    - Azure の場合、ロードバランサーの **api-internal** ルールを削除します。
  - b. パブリックゾーンの **api.\$clustername.\$yourdomain** DNS エントリーを削除します。
2. 外部ロードバランサーを削除します。



重要

以下の手順は、インストーラ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 (IPI) のクラスターでのみ実行できます。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 (UPI) のクラスターの場合は、外部ロードバランサーを手動で削除するか、無効にする必要があります。

- a. ターミナルで、クラスターマシンを一覧表示します。

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

出力例

```
NAME                STATE   TYPE      REGION  ZONE    AGE
lk4pj-master-0      running m4.xlarge us-east-1 us-east-1a 17m
lk4pj-master-1      running m4.xlarge us-east-1 us-east-1b 17m
lk4pj-master-2      running m4.xlarge us-east-1 us-east-1a 17m
lk4pj-worker-us-east-1a-5fzfj running m4.xlarge us-east-1 us-east-1a 15m
lk4pj-worker-us-east-1a-vbghs running m4.xlarge us-east-1 us-east-1a 15m
lk4pj-worker-us-east-1b-zgpzg running m4.xlarge us-east-1 us-east-1b 15m
```

以下の手順で、名前に **master** が含まれるコントロールプレーンマシンを変更します。

- b. 各コントロールプレーンマシンから外部ロードバランサーを削除します。
  - i. コントロールプレーンの **Machine** オブジェクトを編集し、外部ロードバランサーへの参照を削除します。

```
$ oc edit machines -n openshift-machine-api <master_name> 1
```

- 1** 変更するコントロールプレーン (またはマスター) **Machine** オブジェクトの名前を指定します。

- ii. 以下の例でマークが付けられている外部ロードバランサーを記述する行を削除し、オブジェクト仕様を保存し、終了します。

```
...
spec:
  providerSpec:
    value:
      ...
      loadBalancers:
        - name: lk4pj-ext ❶
          type: network ❷
        - name: lk4pj-int
          type: network
```

❶ ❷ この行を削除します。

- iii. 名前に **master** が含まれるマシンにこのプロセスを繰り返します。

2.4.1. Ingress Controller エンドポイント公開スコープの内部への設定

クラスター管理者がクラスターをプライベートに指定せずに新しいクラスターをインストールすると、**scope**が**External**に設定されたデフォルトの Ingress Controller が作成されます。クラスター管理者は、**External** スコープの Ingress Controller を **Internal** に変更できます。

前提条件

- **oc** CLI をインストールしていること。

手順

- **External** スコープの Ingress Controller を **Internal** に変更するには、次のコマンドを入力します。

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService"},"loadBalancer":
{"scope":"Internal"}}}'
```

- Ingress Controller のステータスを確認するには、次のコマンドを入力します。

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

- ステータス状態が **Progressing** の場合は、さらにアクションを実行する必要があるかどうかを示します。たとえば、ステータスの状態によっては、次のコマンドを入力して、サービスを削除する必要があることを示している可能性があります。

```
$ oc -n openshift-ingress delete services/router-default
```

サービスを削除すると、Ingress Operator はサービスを **Internal** として再作成します。

第3章 ベアメタルの設定

ベアメタルホストに OpenShift Container Platform をデプロイする場合、プロビジョニングの前後にホストに変更を加える必要がある場合があります。これには、ホストのハードウェア、ファームウェア、ファームウェアの詳細の検証が含まれます。また、ディスクのフォーマットや、変更可能なファームウェア設定の変更も含まれます。

3.1. BARE METAL OPERATOR について

Bare Metal Operator (BMO) を使用して、クラスター内のベアメタルホストをプロビジョニング、管理、検査します。

BMO は、次の 3 つのリソースを使用してこれらのタスクを完了します。

- **BareMetalHost**
- **HostFirmwareSettings**
- **FirmwareSchema**

BMO は、各ベアメタルホストを **BareMetalHost** カスタムリソース定義のインスタンスにマッピングすることにより、クラスター内の物理ホストのインベントリーを維持します。各 **BareMetalHost** リソースには、ハードウェア、ソフトウェア、およびファームウェアの詳細が含まれています。BMO は、クラスター内のベアメタルホストを継続的に検査して、各 **BareMetalHost** リソースが対応するホストのコンポーネントを正確に詳述していることを確認します。

BMO はまた、**HostFirmwareSettings** リソースと **FirmwareSchema** リソースを使用して、ベアメタルホストのファームウェア仕様を詳述します。

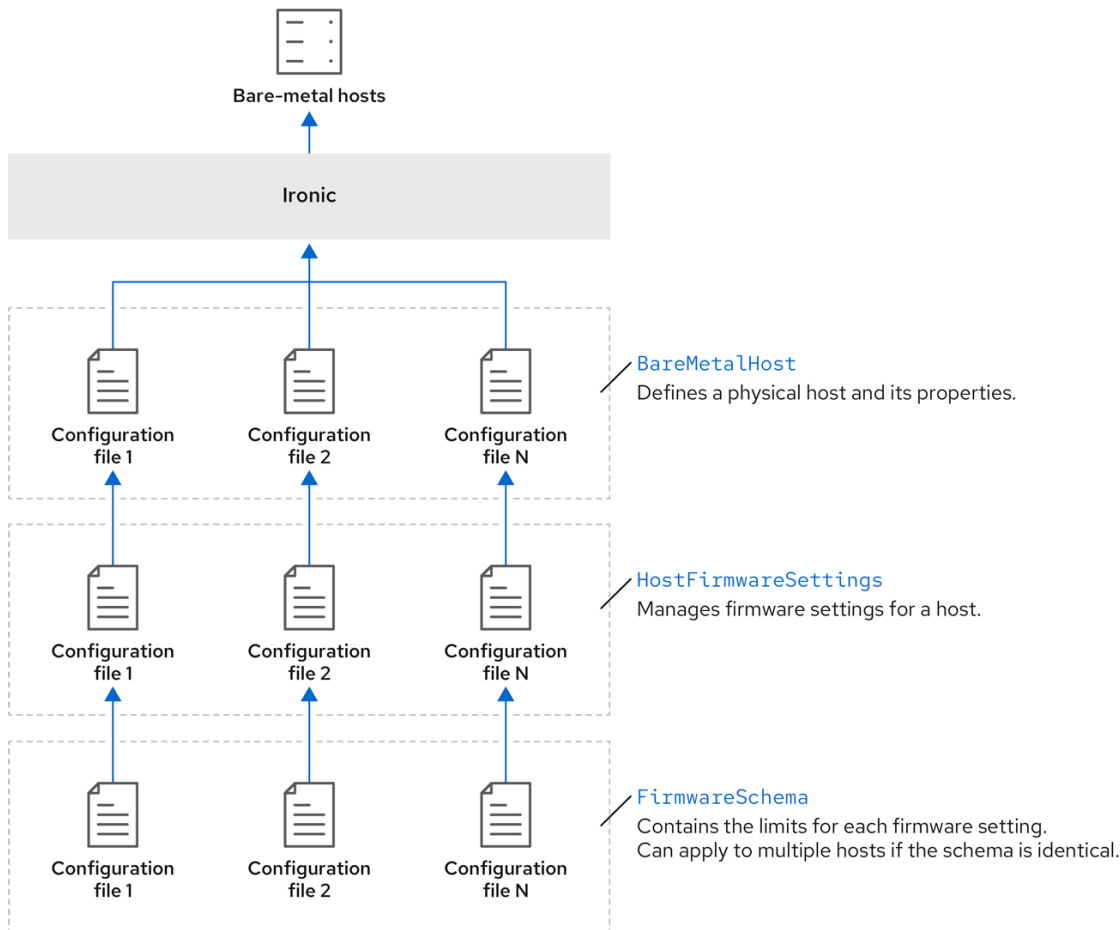
BMO は、Ironic API サービスを使用してクラスター内のベアメタルホストと接続します。Ironic サービスは、ホスト上のベースボード管理コントローラー (BMC) を使用して、マシンと接続します。

BMO を使用して実行できる一般的なタスクには、次のようなものがあります。

- 特定のイメージを使用したクラスターへのベアメタルホストのプロビジョニング
- プロビジョニング前またはプロビジョニング解除後におけるホストのディスクコンテンツのフォーマット
- ホストのオン/オフの切り替え
- ファームウェア設定の変更
- ホストのハードウェア詳細の表示

3.1.1. Bare Metal Operator のアーキテクチャー

Bare Metal Operator (BMO) は、3 つのリソースを使用して、クラスター内のベアメタルホストをプロビジョニング、管理、検査します。次の図は、これらのリソースのアーキテクチャーを示しています。



302_OpenShift_0223

BareMetalHost

BareMetalHost リソースは、物理ホストとそのプロパティを定義します。ベアメタルホストをクラスターにプロビジョニングするときは、そのホストの **BareMetalHost** リソースを定義する必要があります。ホストの継続的な管理のために、**BareMetalHost** の情報を調べたり、この情報を更新したりできます。

BareMetalHost リソースには、次のようなプロビジョニング情報が含まれます。

- オペレーティングシステムのブートイメージやカスタム RAM ディスクなどのデプロイメント仕様
- プロビジョニング状態
- ベースボード管理コントローラー (BMC) アドレス
- 目的の電源状態

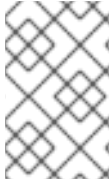
BareMetalHost リソースには、次のようなハードウェア情報が含まれます。

- CPU 数
- NIC の MAC アドレス
- ホストのストレージデバイスのサイズ
- 現在の電源状態

HostFirmwareSettings

HostFirmwareSettings リソースを使用して、ホストのファームウェア設定を取得および管理できます。ホストが **Available** 状態に移行すると、Ironic サービスはホストのファームウェア設定を読み取り、**HostFirmwareSettings** リソースを作成します。**BareMetalHost** リソースと **HostFirmwareSettings** リソースの間には1対1のマッピングがあります。

HostFirmwareSettings リソースを使用して、ホストのファームウェア仕様を調べたり、ホストのファームウェア仕様を更新したりできます。



注記

HostFirmwareSettings リソースの **spec** フィールドを編集するときは、ベンダーファームウェアに固有のスキーマに従う必要があります。このスキーマは、読み取り専用の **FirmwareSchema** リソースで定義されます。

FirmwareSchema

ファームウェア設定は、ハードウェアベンダーやホストモデルによって異なります。**FirmwareSchema** リソースは、各ホストモデル上の各ファームウェア設定のタイプおよび制限が含まれる読み取り専用リソースです。データは、Ironic サービスを使用して BMC から直接取得されます。**FirmwareSchema** リソースを使用すると、**HostFirmwareSettings** リソースの **spec** フィールドに指定できる有効な値を特定できます。

スキーマが同じであれば、**FirmwareSchema** リソースは多くの **BareMetalHost** リソースに適用できます。

関連情報

- [ベアメタルホストをプロビジョニングするための Metal³ API サービス](#)
- [ベアメタルインフラストラクチャーを管理するための Ironic API サービス](#)

3.2. BAREMETALHOST リソースについて

Metal³ で、物理ホストとそのプロパティを定義する **BareMetalHost** リソースの概念が導入されました。**BareMetalHost** リソースには、2つのセクションが含まれます。

1. **BareMetalHost spec**
2. **BareMetalHost status**

3.2.1. BareMetalHost spec

BareMetalHost リソースの **spec** セクションは、ホストの必要な状態を定義します。

表3.1 BareMetalHost spec

パラメーター	説明
automatedCleaningMode	プロビジョニングおよびプロビジョニング解除時の自動クリーニングを有効または無効にするインターフェイス。 disabled に設定すると、自動クリーニングはスキップされます。 metadata に設定すると、自動消去が有効になります。デフォルト設定は metadata です。

パラメーター	説明
bmc: address: credentialsName: disableCertificateVerification:	bmc 設定には、ホスト上のベースボード管理コントローラー (BMC) の接続情報が含まれます。フィールドの詳細は以下のとおりです。 ● address: ホストの BMC コントローラーとの通信用の URL。 ● credentialsName: BMC のユーザー名およびパスワードが含まれるシークレットへの参照。 ● disableCertificateVerification: true に設定されている場合に証明書の検証を省略するブール値。
bootMACAddress	ホストのプロビジョニングに使用する NIC の MAC アドレス。
bootMode	ホストのブートモード。デフォルトは UEFI ですが、BIOS ブートの legacy または UEFISecureBoot に設定することもできます。
consumerRef	ホストを使用している別のリソースへの参照。別のリソースが現在ホストを使用していない場合は、空になることがあります。たとえば、 machine-api がホストを使用している場合に、 Machine リソースがホストを使用する場合があります。
description	ホストの特定に役立つ、人間が提供した文字列。
externallyProvisioned	ホストのプロビジョニングとプロビジョニング解除が外部で管理されるかどうかを示すブール値。設定される場合: ● 電源ステータスは、オンラインフィールドを使用して引き続き管理できます。 ● ハードウェアインベントリは監視されますが、プロビジョニング操作やプロビジョニング解除操作はホストで実行されません。

パラメーター	説明
firmware	ベアメタルホストの BIOS 設定に関する情報が含まれます。現在、firmware は、iRMC、iDRAC、iLO4、および iLO5 BMC でのみサポートされます。サブフィールドは以下のとおりです。 ● simultaneousMultithreadingEnabled: 単一の物理プロセッサコアが複数の論理プロセッサとして表示されるのを許可します。有効な設定は true または false です。 ● sriovEnabled: SR-IOV のサポートにより、ハイパーバイザーが PCI-express デバイスの仮想インスタンスを作成できるようになり、パフォーマンスが向上する可能性があります。有効な設定は true または false です。 ● virtualizationEnabled: プラットフォームハードウェアの仮想化をサポートします。有効な設定は true または false です。
image: url: checksum: checksumType: format:	image 設定には、ホストにデプロイされるイメージの詳細が保持されます。Ironic にはイメージフィールドが必要です。ただし、externallyProvisioned 設定が true に設定され、外部管理に電源管理が必要な場合は、フィールドは空にすることができます。フィールドの詳細は以下のとおりです。 ● URL: ホストにデプロイするイメージの URL。 ● checksum: 実際のチェックサム、または image.url のイメージのチェックサムが含まれるファイルへの URL。 ● checksumType: チェックサムアルゴリズムを指定できます。現時点で image.checksumType は md5、sha256、および sha512 のみをサポートしています。デフォルトのチェックサムタイプは md5 です。 ● format: これはイメージのディスク形式です。raw、qcow2、vdi、vmdk、live-iso のいずれか、未設定のままにすることができます。これを raw に設定すると、そのイメージの Ironic エージェントでの raw イメージのストリーミングが有効になります。これを live-iso に設定すると、iso イメージをディスクにデプロイせずにライブブートが可能になり、checksum フィールドは無視されます。

パラメーター	説明
networkData	ネットワーク設定データおよびその namespace が含まれるシークレットへの参照。したがって、ホストが起動してネットワークをセットアップする前にホストに接続することができます。
online	ホストの電源を入れる (true) かオフにする (false) かを示すブール値。この値を変更すると、物理ホストの電源状態に変更が加えられます。
raid: hardwareRAIDVolumes: softwareRAIDVolumes:	(オプション) ベアメタルホストの RAID 設定に関する情報が含まれます。指定しない場合は、現在の設定を保持します。  注記 OpenShift Container Platform 4.10 は、iRMC プロトコルのみを使用して BMC のハードウェア RAID をサポートします。OpenShift Container Platform 4.10 は、ソフトウェア RAID をサポートしていません。 次の構成設定を参照してください。 ● hardwareRAIDVolumes: ハードウェア RAID の論理ドライブの一覧が含まれ、ハードウェア RAID で必要なボリューム設定を定義します。rootDeviceHints を指定しない場合、最初のボリュームがルートボリュームになります。サブフィールドは以下のとおりです。 ○ level: 論理ドライブの RAID レベル。0、1、2、5、6、1+0、5+0、6+0 のレベルがサポートされます。 ○ name: 文字列としてのボリュームの名前。サーバー内で一意である必要があります。指定されていない場合、ボリューム名は自動生成されます。 ○ numberOfPhysicalDisks: 論理ドライブに使用する物理ドライブの数 (整数)。デフォルトは、特定の RAID レベルに必要なディスクドライブの最小数です。 ○ physicalDisks: 物理ディスクドライブの名前の一覧です (文字列)。これはオプションのフィールドです。指定した場合、controller フィールドも指定する必要があります。 ○ controller: (オプション) ハードウェア RAID ボリュームで使用する RAID コントローラーの名前 (文字列)。 ○ rotational: true に設定すると、回転

パラメーター	説明
	ディスクを用いるドライブのみが選択されます。false に設定すると、ソリッドステートドライブと NVMe ドライブのみが選択されます。設定されていない場合は、任意のドライブの種類を選択します (デフォルト動作)。 ○ sizeGibibytes: 作成する論理ドライブのサイズ (GiB 単位の整数)。指定がない場合や 0 に設定すると、論理ドライブ用に物理ドライブの最大容量が使用されます。 ● softwareRAIDVolumes: OpenShift Container Platform 4.10 は、ソフトウェア RAID をサポートしていません。以下の情報は参考用です。この設定には、ソフトウェア RAID の論理ディスクのリストが含まれています。rootDeviceHints を指定しない場合、最初のボリュームがルートボリュームになります。HardwareRAIDVolumes を設定すると、この項目は無効になります。ソフトウェア RAID は常に削除されます。作成されるソフトウェア RAID デバイスの数は、1 または 2 である必要があります。ソフトウェア RAID デバイスが 1 つしかない場合は、RAID-1 にする必要があります。2 つの RAID デバイスがある場合は、1 番目のデバイスを RAID-1 にする必要があります。また、2 番目のデバイスの RAID レベルは 0、1、または 1+0 に設定できます。最初の RAID デバイスがデプロイメントデバイスになります。したがって、RAID-1 を強制すると、デバイスに障害が発生した場合のノードが起動しないリスクが軽減されます。softwareRAIDVolume フィールドは、ソフトウェア RAID のボリュームの必要な設定を定義します。サブフィールドは以下のとおりです。 ○ level: 論理ドライブの RAID レベル。0、1、1+0 のレベルがサポートされます。 ○ physicalDisks: デバイスのヒントの一覧。アイテム数は、2 以上である必要があります。 ○ sizeGibibytes: 作成される論理ディスクドライブのサイズ (GiB 単位の整数)。指定がない場合や 0 に設定すると、論理ドライブ用に物理ドライブの最大容量が使用されます。 hardwareRAIDVolume を空のスライスとして設定すると、ハードウェア RAID 設定を消去できます。以下に例を示します。 <pre>spec: raid: hardwareRAIDVolume: []</pre> ドライバーが RAID に対応していないことを示すエラーメッセージが表示された場合

パラメーター	説明
	は、<code>raid</code>、<code>hardwareRAIDVolumes</code> または <code>softwareRAIDVolumes</code> を <code>nil</code> に設定します。ホストに RAID コントローラーがあることを確認する必要があります。

パラメーター	説明
<pre> rootDeviceHints: deviceName: hctl: model: vendor: serialNumber: minSizeGigabytes: wwn: wwnWithExtension: wwnVendorExtension: rotational: </pre>	rootDeviceHints パラメーターを使用すると、特定のデバイスへの RHCOS イメージのプロビジョニングが可能になります。これは、検出順にデバイスを検査し、検出された値をヒントの値と比較します。ヒントの値と一致する最初に検出されたデバイスが使用されます。設定では複数のヒントを組み合わせることができますが、デバイスが選択されるには、デバイスがすべてのヒントと一致する必要があります。フィールドの詳細は以下のとおりです。 ● deviceName: /dev/vda などの Linux デバイス名が含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● hctl: 0:0:0:0 などの SCSI バスアドレスが含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● model: ベンダー固有のデバイス識別子が含まれる文字列。ヒントは、実際の値のサブ文字列になります。 ● vendor: デバイスのベンダーまたは製造元の名前が含まれる文字列。ヒントは、実際の値のサブ文字列になります。 ● serialNumber: デバイスのシリアル番号が含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● minSizeGigabytes: デバイスの最小サイズを表す整数 (ギガバイト単位)。 ● wwn: 一意のストレージ ID が含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● wwnWithExtension: ベンダー拡張が追加された一意のストレージ ID が含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● wwnVendorExtension: 一意のベンダーストレージ ID が含まれる文字列。ヒントは、実際の値と完全に一致する必要があります。 ● rotational: デバイスが回転ディスクを用いる (true) か、そうでないか (false) を示すブール値。

3.2.2. BareMetalHost status

BareMetalHost status は、ホストの現在の状態を表し、テスト済みの認証情報、現在のハードウェアの詳細などの情報が含まれます。

表3.2 BareMetalHost status

パラメーター	説明
goodCredentials	シークレットおよびその namespace の参照で、システムが動作中と検証できるベースボード管理コントローラー (BMC) 認証情報のセットが保持されています。
errorMessage	プロビジョニングバックエンドが報告する最後のエラーの詳細 (ある場合)。
errorType	ホストがエラー状態になった原因となった問題のクラスを示します。エラータイプは以下のとおりです。 ● provisioned registration error: コントローラーがプロビジョニング済みのホストを再登録できない場合に発生します。 ● registration error: コントローラーがホストのベースボード管理コントローラーに接続できない場合に発生します。 ● inspection error: ホストからハードウェア詳細の取得を試みて失敗した場合に発生します。 ● preparation error: クリーニングに失敗した場合に発生します。 ● provisioning error: コントローラーがホストのプロビジョニングまたはプロビジョニング解除に失敗した場合に発生します。 ● power management error: コントローラーがホストの電源状態を変更できない場合に発生します。 ● detach error: コントローラーがホストをプロビジョナーからデタッチできない場合に発生します。

パラメーター	説明
<pre>hardware: cpu arch: model: clockMegahertz: flags: count:</pre>	hardware.cpu フィールドは、システム内の CPU の詳細を示します。フィールドには以下が含まれます。 ● arch: CPU のアーキテクチャー。 ● model: CPU モデル (文字列)。 ● clockMegahertz: CPU の速度 (MHz 単位)。 ● flags: CPU フラグの一覧。たとえば、'mmx'、'sse'、'sse2'、'vmx'などです。 ● count: システムで利用可能な CPU の数。
<pre>hardware: firmware:</pre>	BIOS ファームウェア情報が含まれます。たとえば、ハードウェアベンダーおよびバージョンなどです。
<pre>hardware: nics: - ip: name: mac: speedGbps: vlans: vlanId: pxe:</pre>	hardware.nics フィールドには、ホストのネットワークインターフェイスの一覧が含まれます。フィールドには以下が含まれます。 ● ip: NIC の IP アドレス (検出エージェントの実行時に IP アドレスが割り当てられている場合)。 ● name: ネットワークデバイスを識別する文字列。例:nic-1 ● mac: NIC の MAC アドレス。 ● speedGbps: デバイスの速度 (Gbps 単位)。 ● vlans: この NIC で利用可能な VLAN をすべて保持するリスト。 ● vlanid: タグ付けされていない VLAN ID。 ● pxe: NIC が PXE を使用して起動できるかどうか。
<pre>hardware: ramMebibytes:</pre>	ホストのメモリー容量 (MiB 単位)。

パラメーター	説明
<pre>hardware: storage: - name: rotational: sizeBytes: serialNumber:</pre>	hardware.storage フィールドには、ホストで利用可能なストレージデバイスの一覧が含まれます。フィールドには以下が含まれます。 ● name: ストレージデバイスを識別する文字列。たとえば、disk 1 (boot) などです。 ● rotational: ディスクが回転ディスクを用いるかどうかを示します。true または false のいずれかを返します。 ● sizeBytes: ストレージデバイスのサイズ。 ● serialNumber: デバイスのシリアル番号。
<pre>hardware: systemVendor: manufacturer: productName: serialNumber:</pre>	ホストの 製造元、製品名、および シリアル番号 に関する情報が含まれます。
lastUpdated	ホストのステータスの最終更新時のタイムスタンプ。
operationalStatus	サーバーのステータス。ステータスは以下のいずれかになります。 ● OK: ホストの詳細がすべて認識され、正しく設定され、機能し、管理可能であることを示します。 ● discovered: ホストの詳細の一部が正常に動作していないか、欠落しているかのいずれかを意味します。たとえば、BMC アドレスは認識されているが、ログイン認証情報が認識されていない。 ● error: システムで回復不能なエラーが検出されたことを示します。詳細は、status セクションの errorMessage フィールドを参照してください。 ● delayed: 複数ホストの同時プロビジョニングを制限するために、プロビジョニングが遅延していることを示します。 ● detached: ホストが unmanaged として識別されていることを示します。
poweredOn	ホストの電源が入っているかどうかを示すブール値。

provisioning: state: パラメーター	provisioning フィールドには、ホストへのイメージのデプロイに関連する値が含まれます。サブフィールドには以下が含まれます。 説明
id: image: raid: firmware: rootDeviceHints:	state: 進行中のプロビジョニング操作の現在の状態。状態には、以下が含まれます。 <空の文字列>: 現時点でプロビジョニングは行われていません。 unmanaged: ホストを登録するのに十分な情報が利用できません。 registering: エージェントはホストの BMC の詳細を確認しています。 match profile: エージェントは、ホストで検出されたハードウェア詳細と既知のプロファイルを比較しています。 available: ホストはプロビジョニングに使用できます。この状態は、以前は ready として知られていました。 preparing: 既存の設定は削除され、新しい設定がホストに設定されます。 provisioning: プロビジョナーはイメージをホストのストレージに書き込んでいます。 provisioned: プロビジョナーはイメージをホストのストレージに書き込みました。 externally provisioned: Meta³ は、ホスト上のイメージを管理しません。 deprovisioning: プロビジョナーは、ホストのストレージからイメージを消去しています。 inspecting: エージェントはホストのハードウェア情報を収集しています。 deleting: エージェントはクラスターから を削除しています。 id: 基礎となるプロビジョニングツールのサービスの一意識別子。 image: 直近ホストにプロビジョニングされたイメージ。 raid: 最近設定したハードウェアまたはソフトウェア RAID ボリュームの一覧。 firmware: ベアメタルサーバーの BIOS 設定。 rootDeviceHints: 直近のプロビジョニング操作に使用されたルートデバイス選択の手順。

パラメーター	説明
triedCredentials	プロビジョニングバックエンドに送信された BMC 認証情報の最後のセットを保持するシークレットおよびその namespace への参照。

3.3. BAREMETALHOST リソースの取得

BareMetalHost リソースには、物理ホストのプロパティが含まれます。物理ホストのプロパティをチェックするには、その **BareMetalHost** リソースを取得する必要があります。

手順

1. **BareMetalHost** リソースの一覧を取得します。

```
$ oc get bmh -n openshift-machine-api -o yaml
```



注記

oc get コマンドで、**bmh** の長い形式として、**baremetalhost**を使用できます。

2. ホストの一覧を取得します。

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

3. 特定のホストの **BareMetalHost** リソースを取得します。

```
$ oc get bmh <host_name> -n openshift-machine-api -o yaml
```

ここで、**<host_name>** はホストの名前です。

出力例

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  creationTimestamp: "2022-06-16T10:48:33Z"
  finalizers:
    - baremetalhost.metal3.io
  generation: 2
  name: openshift-worker-0
  namespace: openshift-machine-api
  resourceVersion: "30099"
  uid: 1513ae9b-e092-409d-be1b-ad08edeb1271
spec:
  automatedCleaningMode: metadata
  bmc:
    address: redfish://10.46.61.19:443/redfish/v1/Systems/1
    credentialsName: openshift-worker-0-bmc-secret
    disableCertificateVerification: true
    bootMACAddress: 48:df:37:c7:f7:b0
```

```
bootMode: UEFI
consumerRef:
  apiVersion: machine.openshift.io/v1beta1
  kind: Machine
  name: ocp-edge-958fk-worker-0-nrfcg
  namespace: openshift-machine-api
customDeploy:
  method: install_coreos
hardwareProfile: unknown
online: true
rootDeviceHints:
  deviceName: /dev/sda
userData:
  name: worker-user-data-managed
  namespace: openshift-machine-api
status:
  errorCount: 0
  errorMessage: ""
  goodCredentials:
    credentials:
      name: openshift-worker-0-bmc-secret
      namespace: openshift-machine-api
    credentialsVersion: "16120"
  hardware:
    cpu:
      arch: x86_64
      clockMegahertz: 2300
      count: 64
      flags:
        - 3dnowprefetch
        - abm
        - acpi
        - adx
        - aes
      model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
    firmware:
      bios:
        date: 10/26/2020
        vendor: HPE
        version: U30
      hostname: openshift-worker-0
    nics:
      - mac: 48:df:37:c7:f7:b3
        model: 0x8086 0x1572
        name: ens1f3
    ramMebibytes: 262144
    storage:
      - hctl: "0:0:0:0"
        model: VK000960GWTTB
        name: /dev/sda
        sizeBytes: 960197124096
        type: SSD
        vendor: ATA
    systemVendor:
      manufacturer: HPE
      productName: ProLiant DL380 Gen10 (868703-B21)
```

```

serialNumber: CZ200606M3
hardwareProfile: unknown
lastUpdated: "2022-06-16T11:41:42Z"
operationalStatus: OK
poweredOn: true
provisioning:
  ID: 217baa14-cfcf-4196-b764-744e184a3413
  bootMode: UEFI
  customDeploy:
    method: install_coreos
  image:
    url: ""
  raid:
    hardwareRAIDVolumes: null
    softwareRAIDVolumes: []
  rootDeviceHints:
    deviceName: /dev/sda
    state: provisioned
  triedCredentials:
    credentials:
      name: openshift-worker-0-bmc-secret
      namespace: openshift-machine-api
    credentialsVersion: "16120"

```

3.4. HOSTFIRMWARESETTINGS リソースについて

HostFirmwareSettings リソースを使用して、ホストの BIOS 設定を取得および管理できます。ホストが **Available** 状態に移行すると、Ironic はホストの BIOS 設定を読み取り、**HostFirmwareSettings** リソースを作成します。リソースには、ベースボード管理コントローラー (BMC) から返される完全な BIOS 設定が含まれます。**BareMetalHost** リソースの **firmware** フィールドは、ベンダーに依存しない 3 つのフィールドを返しますが、**HostFirmwareSettings** リソースは、通常ホストごとにベンダー固有のフィールドの多数の BIOS 設定で設定されます。

HostFirmwareSettings リソースには、以下の 2 つのセクションが含まれます。

1. **HostFirmwareSettings** spec
2. **HostFirmwareSettings** status

3.4.1. HostFirmwareSettings spec

HostFirmwareSettings リソースの **spec** セクションは、ホストの BIOS の必要な状態を定義し、デフォルトでは空です。Ironic は **spec.settings** セクションの設定を使用して、ホストが **Preparing** 状態の場合、ベースボード管理コントローラー (BMC) を更新します。**FirmwareSchema** リソースを使用して、無効な名前と値のペアをホストに送信しないようにします。詳細は、**FirmwareSchema** リソースについてを参照してください。

例

```

spec:
  settings:
    ProcTurboMode: Disabled ❶

```

- 1 前述の例では、**spec.settings** セクションには、**ProcTurboMode** BIOS 設定を **Disabled** に設定する名前/値のペアが含まれます。



注記

status セクションに一覧表示される整数パラメーターは文字列として表示されます。たとえば、**"1"** と表示されます。**spec.settings** セクションで整数を設定する場合、値は引用符なしの整数として設定する必要があります。たとえば、**1** と設定します。

3.4.2. HostFirmwareSettings status

status は、ホストの BIOS の現在の状態を表します。

表3.3 HostFirmwareSettings

パラメーター	説明
status: conditions: - lastTransitionTime: message: observedGeneration: reason: status: type:	conditions フィールドには、状態変更の一覧が含まれます。サブフィールドには以下が含まれます。 ● lastTransitionTime: 状態が最後に変更した時刻。 ● message: 状態変更の説明。 ● observedGeneration: status の現在の生成。metadata.generation とこのフィールドが同じでない場合には、status.conditions が古い可能性があります。 ● reason: 状態変更の理由。 ● status: 状態の変更のステータス。ステータスは True、False、または Unknown です。 ● type: 状態変更のタイプ。タイプは Valid および ChangeDetected です。
status: schema: name: namespace: lastUpdated:	ファームウェア設定の FirmwareSchema。フィールドには以下が含まれます。 ● name: スキーマを参照する名前または一意の識別子。 ● namespace: スキーマが保存される namespace。 ● lastUpdated: リソースが最後に更新された時刻。
status: settings:	settings フィールドには、ホストの現在の BIOS 設定の名前と値のペアのリストが含まれます。

3.5. HOSTFIRMWARESETTINGS リソースの取得

HostFirmwareSettings リソースには、物理ホストのベンダー固有の BIOS プロパティーが含まれます。物理ホストの BIOS プロパティーをチェックするには、その**HostFirmwareSettings** リソースを取得する必要があります。

手順

1. **HostFirmwareSettings** リソースの詳細な一覧を取得します。

```
$ oc get hfs -n openshift-machine-api -o yaml
```



注記

oc get コマンドで、**hfs** の長い形式として、**hostfirmwaresettings**を使用できます。

2. **HostFirmwareSettings** リソースの一覧を取得します。

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

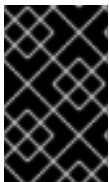
3. 特定のホストの **HostFirmwareSettings** リソースを取得します。

```
$ oc get hfs <host_name> -n openshift-machine-api -o yaml
```

ここで、**<host_name>** はホストの名前です。

3.6. HOSTFIRMWARESETTINGS リソースの編集

プロビジョニングされたホストの **HostFirmwareSettings** を編集できます。



重要

読み取り専用の値を除き、ホストが **プロビジョニング** された状態にある場合にのみ、ホストを編集できます。**外部からプロビジョニング** された状態のホストは編集できません。

手順

1. **HostFirmwareSettings** リソースの一覧を取得します。

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

2. ホストの **HostFirmwareSettings** リソースを編集します。

```
$ oc edit hfs <host_name> -n openshift-machine-api
```

ここで、**<host_name>** はプロビジョニングされたホストの名前です。**HostFirmwareSettings** リソースは、ターミナルのデフォルトエディターで開きます。

3. **spec.settings** セクションに、名前と値のペアを追加します。

例

```
spec:
  settings:
    name: value ❶
```

- ❶ **FirmwareSchema** リソースを使用して、ホストで利用可能な設定を特定します。読み取り専用の値は設定できません。

4. 変更を保存し、エディターを終了します。

5. ホストのマシン名を取得します。

```
$ oc get bmh <host_name> -n openshift-machine name
```

ここで、**<host_name>** はホストの名前です。マシン名は **CONSUMER** フィールドの下に表示されます。

6. マシンにアノテーションを付け、マシンセットから削除します。

```
$ oc annotate machine <machine_name> machine.openshift.io/cluster-api-delete-machine=yes -n openshift-machine-api
```

ここで、**<machine_name>** は削除するマシンの名前です。

7. ノードの一覧を取得し、ワーカーノードの数をカウントします。

```
$ oc get nodes
```

8. マシンセットを取得します。

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

9. マシンセットをスケーリングします。

```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n-1>
```

ここで、**<machineset_name>** はマシンセットの名前で、**<n-1>** は減少させたワーカーノードの数です。

10. ホストが **Available** の状態になったら、Machineset をスケールアップして、**HostFirmwareSettings** リソースの変更を反映させます。

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

ここで、**<machineset_name>** はマシンセットの名前で、**<n>** はワーカーノードの数です。

3.7. HOSTFIRMWARE SETTINGS リソースが有効であることの確認

ユーザーが **spec.settings** セクションを編集して **HostFirmwareSetting** (HFS) リソースに変更を加えると、Bare Metal Operator (BMO) は読み取り専用リソースである **FirmwareSchema** リソースに対して変更を検証します。この設定が無効な場合、BMO は **status.Condition** 設定の **Type** の値を **False** に

設定し、イベントを生成して HFS リソースに保存します。以下の手順を使用して、リソースが有効であることを確認します。

手順

- 1. **HostFirmwareSetting** リソースの一覧を取得します。

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

- 2. 特定のホストの **HostFirmwareSettings** リソースが有効であることを確認します。

```
$ oc describe hfs <host_name> -n openshift-machine-api
```

ここで、<host_name> はホストの名前です。

出力例

```
Events:
  Type    Reason          Age    From                                     Message
  ----    -
  Normal  ValidationFailed 2m49s  metal3-hostfirmwaresettings-controller Invalid BIOS
setting: Setting ProcTurboMode is invalid, unknown enumeration value - Foo
```



重要

応答が **ValidationFailed** を返す場合、リソース設定にエラーがあり、**FirmwareSchema** リソースに準拠するよう値を更新する必要があります。

3.8. FIRMWARESCHEMA リソースについて

BIOS 設定は、ハードウェアベンダーやホストモデルによって異なります。**FirmwareSchema** リソースは、各ホストモデル上の各 BIOS 設定のタイプおよび制限が含まれる読み取り専用リソースです。データは BMC から IroniC に直接取得されます。**FirmwareSchema** を使用すると、**HostFirmwareSettings** リソースの **spec** フィールドに指定できる有効な値を特定できます。**FirmwareSchema** リソースには、その設定および制限から派生する一意の識別子があります。同じホストモデルは同じ **FirmwareSchema** 識別子を使用します。**HostFirmwareSettings** の複数のインスタンスが同じ **FirmwareSchema** を使用する可能性が高いです。

表3.4 FirmwareSchema 仕様

パラメーター	説明
<pre> <BIOS_setting_name> attribute_type: allowable_values: lower_bound: upper_bound: min_length: max_length: read_only: unique: </pre>	spec は、BIOS 設定名と設定の制限で設定される単純なマップです。フィールドには以下が含まれます。 ● attribute_type: 設定のタイプ。サポートされるタイプは以下のとおりです。 ○ 列挙 ○ 整数 ○ 文字列 ○ Boolean ● allowable_values: attribute_type が Enumeration の場合の、許可される値のリスト。 ● lower_bound: attribute_type が 整数 の場合の下限值。 ● upper_bound: attribute_type が 整数 の場合の上限値。 ● min_length: attribute_type が 文字列 の場合に、値が取ることのできる最も短い文字列の長さ。 ● max_length: attribute_type が 文字列 の場合に、値が取ることのできる最も長い文字列の長さ。 ● read_only: 設定は読み取り専用で、変更することはできません。 ● unique: 設定はこのホストに固有のものです。

3.9. FIRMWARESCHEMA リソースの取得

各ベンダーの各ホストモデルの BIOS 設定は、それぞれ異なります。**HostFirmwareSettings** リソースの **spec** セクションを編集する際に、設定する名前/値のペアはそのホストのファームウェアスキーマに準拠している必要があります。有効な名前と値のペアを設定するには、ホストの **FirmwareSchema** を取得して確認します。

手順

1. **FirmwareSchema** リソースインスタンスの一覧を取得するには、以下を実行します。

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

2. 特定の **FirmwareSchema** インスタンスを取得するには、以下を実行します。

```
$ oc get firmwareschema <instance_name> -n openshift-machine-api -o yaml
```

ここで、<instance_name> は、**HostFirmwareSettings** リソース (表 3 を参照) に記載されているスキーマインスタンスの名前です。

第4章 インストール後のマシン設定タスク

OpenShift Container Platform ノードで実行しているオペレーティングシステムに変更を加える必要がある場合があります。これには、ネットワークタイムサービスの設定変更、カーネル引数の追加、特定の方法でのジャーナルの設定などが含まれます。

いくつかの特殊な機能のほかに、OpenShift Container Platform ノードのオペレーティングシステムへの変更のほとんどは、Machine Config Operator によって管理される **MachineConfig** オブジェクトというオブジェクトを作成することで実行できます。

このセクションのタスクでは、Machine Config Operator の機能を使用して OpenShift Container Platform ノードでオペレーティングシステム機能を設定する方法を説明します。

4.1. MACHINE CONFIG OPERATOR について

4.1.1. Machine Config Operator

目的

Machine Config Operator は、カーネルと kubelet 間のすべてのものを含め、ベースオペレーティングシステムおよびコンテナランタイムの設定および更新を管理し、適用します。

以下の 4 つのコンポーネントがあります。

- **machine-config-server**: クラスタに参加する新規マシンに Ignition 設定を提供します。
- **machine-config-controller**: マシンのアップグレードを **MachineConfig** オブジェクトで定義される必要な設定に調整します。マシンセットのアップグレードを個別に制御するオプションが提供されます。
- **machine-config-daemon**: 更新時に新規のマシン設定を適用します。マシンの状態を要求されたマシン設定に対して検証し、確認します。
- **machine-config**: インストール時のマシン設定の完全なソース、初回の起動、およびマシンの更新を提供します。

重要

現在、マシン設定サーバーエンドポイントをブロックまたは制限する方法はサポートされていません。マシン設定サーバーは、既存の設定または状態を持たない新しくプロビジョニングされたマシンが設定を取得できるように、ネットワークに公開する必要があります。このモデルでは、信頼のルートは証明書署名要求 (CSR) エンドポイントであり、kubelet がクラスタに参加するための承認のために証明書署名要求を送信する場所です。このため、シークレットや証明書などの機密情報を配布するためにマシン設定を使用しないでください。

マシン設定サーバーエンドポイント、ポート 22623 および 22624 がベアメタルシナリオで確実に保護されるようにするには、顧客は適切なネットワークポリシーを設定する必要があります。

関連情報

- [OpenShift SDN ネットワークプラグインについて](#)

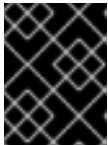
プロジェクト

[openshift-machine-config-operator](#)

4.1.2. マシン設定の概要

Machine Config Operator (MCO) は systemd、CRI-O、Kubelet、カーネル、ネットワークマネージャーその他のシステム機能への更新を管理します。また、これはホストに設定ファイルを書き込むことができる **MachineConfig** CRD を提供します ([machine-config-operator](#) を参照)。MCO の機能や、これが他のコンポーネントとどのように対話するかを理解することは、詳細なシステムレベルの変更を OpenShift Container Platform クラスタに加える上で重要です。以下は、MCO、マシン設定、およびそれらの使用方法について知っておく必要のある点です。

- マシン設定は、OpenShift Container Platform ノードのプールを表す各システムのオペレーティングシステムのファイルまたはサービスに特定の変更を加えることができます。
- MCO はマシンのプールのオペレーティングシステムに変更を適用します。すべての OpenShift Container Platform クラスタについては、ワーカーおよびコントロールプレーンノードプールから始まります。ロールラベルを追加することで、ノードのカスタムプールを設定できます。たとえば、アプリケーションが必要とする特定のハードウェア機能が含まれるワーカーノードのカスタムプールを設定できます。ただし、本セクションの例では、デフォルトのプールタイプの変更に重点を置いています。



重要

ノードには、**master** または **worker** などの複数のラベルを適用できますが、ノードを **単一の** マシン設定プールのメンバーにすることもできます。

- 一部のマシン設定は、OpenShift Container Platform がディスクにインストールされる前に行われる必要があります。ほとんどの場合、これは、インストール後のマシン設定として実行されるのではなく、OpenShift Container Platform インストーラプロセスに直接挿入されるマシン設定を作成して実行できます。他の場合に、ノードごとの個別 IP アドレスの設定や高度なディスクのパーティション設定などを行うには、OpenShift Container Platform インストーラの起動時にカーネル引数を渡すベアメタルのインストールを実行する必要がある場合があります。
- MCO はマシン設定で設定される項目を管理します。MCO が競合するファイルを管理することを明示的に指示されない限り、システムに手動で行う変更は MCO によって上書きされることはありません。つまり、MCO は要求される特定の更新のみを行い、ノード全体に対する制御を要求しません。
- ノードの手動による変更は推奨されません。ノードの使用を中止して新規ノードを起動する必要がある場合は、それらの直接的な変更は失われます。
- MCO は、**/etc** および **/var** ディレクトリーのファイルに書き込みを行う場合にのみサポートされます。ただし、これらの領域のいずれかにシンボリックリンクを指定して書き込み可能なディレクトリーへのシンボリックリンクもあります。**/opt** および **/usr/local** ディレクトリーが例になります。
- Ignition は MachineConfig で使用される設定形式です。詳細は、[Ignition 設定仕様 v3.2.0](#) を参照してください。
- Ignition 設定は OpenShift Container Platform のインストール時に直接提供でき、MCO が Ignition 設定を提供するのと同じ方法でフォーマットできますが、MCO では元の Ignition 設定を確認する方法がありません。そのため、それらをデプロイする前に Ignition 設定をマシン設定にラップする必要があります。
- MCO で管理されるファイルが MCO 外で変更されると、Machine Config Daemon (MCD) はノードを **degraded** として設定します。これは問題のあるファイルを上書きしませんが、継続して **degraded** 状態で動作します。

- マシンの設定を使用する主な理由として、これは OpenShift Container Platform クラスターのプールに対して新規ノードを起動する際に適用されます。**machine-api-operator** は新規マシンをプロビジョニングし、MCO がこれを設定します。

MCO は [Ignition](#) を設定形式として使用します。OpenShift Container Platform バージョン 4.6 では、Ignition 設定仕様のバージョン 2 から 3 に移行しています。

4.1.2.1. マシン設定で変更できる設定

MCO で変更できるコンポーネントの種類には、以下が含まれます。

- config:** ignition 設定オブジェクト ([Ignition 設定仕様](#) を参照してください) を作成し、以下を含む OpenShift Container Platform マシン上でのファイル、systemd サービスおよびその他の機能の変更などを実行できます。
  - Configuration files:** `/var` または `/etc` ディレクトリーでファイルを作成するか、上書きします。
  - systemd units:** systemd サービスを作成し、そのステータスを設定するか、追加設定により既存の systemd サービスに追加します。
  - users and groups:** インストール後に `passwd` セクションで SSH キーを変更します。



重要

- マシン設定を使用した SSH キーの変更は、**core** ユーザーにのみサポートされています。
- マシン設定を使用した新しいユーザーの追加はサポートされていません。

- KernelArguments:** OpenShift Container Platform ノードの起動時に、引数をカーネルコマンドラインに追加します。
- kernelType:** オプションで、標準カーネルの代わりに使用する標準以外のカーネルを特定します。(RAN の) RT カーネルを使用するには、**realtime** を使用します。これは一部のプラットフォームでのみサポートされます。
- fips:** [FIPS](#) モードを有効にします。FIPS は、インストール後の手順ではなく、インストール時の設定で設定される必要があります。



重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL での FIPS モードの設定の詳細は、[FIPS モードでのシステムのインストール](#) を参照してください。プロセス暗号化ライブラリーでの FIPS 検証済みまたはモジュールの使用は、**x86_64** アーキテクチャーでの OpenShift Container Platform デプロイメントでのみサポートされます。

- extensions:** 事前にパッケージ化されたソフトウェアを追加して RHCOS 機能を拡張します。この機能については、利用可能な拡張機能には [usbguard](#) およびカーネルモジュールが含まれます。

- **カスタムリソース (ContainerRuntime および Kubelet 用)**: マシン設定外で、MCO は CRI-O コンテナランタイムの設定 (**ContainerRuntime** CR) および Kubelet サービス (**Kubelet** CR) を変更するために2つの特殊なカスタムリソースを管理します。

MCO は、OpenShift Container Platform ノードでオペレーティングシステムコンポーネントを変更できる唯一の Operator という訳ではありません。他の Operator もオペレーティングシステムレベルの機能を変更できます。1つの例として、Node Tuning Operator を使用して、Tuned デーモンプロファイルを使用したノードレベルのチューニングを実行できます。

インストール後に実行可能な MCO 設定のタスクは、以下の手順に記載されています。OpenShift Container Platform のインストール時またはインストール前に実行する必要があるシステム設定タスクについては、RHCOS ベアメタルのインストールについての説明を参照してください。

ノードの設定が、現在適用されているマシン設定で指定されているものと完全に一致しない場合があります。この状態は **設定ドリフト** と呼ばれます。Machine Config Daemon (MCD) は、ノードの設定ドリフトを定期的にチェックします。MCD が設定のドリフトを検出した場合は、管理者がノード設定を修正するまで、MCO はノードを **劣化** とマークします。劣化したノードはオンラインで動作していますが、更新できません。設定ドリフトの詳細は、**Understanding configuration drift detection** を参照してください。

4.1.2.2. プロジェクト

詳細は、[openshift-machine-config-operator](#) GitHub サイトを参照してください。

4.1.3. 設定ドリフト検出について

ノードのディスク上の状態がマシン設定で設定される内容と異なる場合があります。これは、**設定ドリフト** と呼ばれます。たとえば、クラスター管理者は、マシン設定で設定されたファイル、systemd ユニットファイル、またはファイルパーミッションを手動で変更する場合があります。これにより、設定のドリフトが発生します。設定ドリフトにより、Machine Config Pool のノード間で問題が発生したり、マシン設定が更新されると問題が発生したりする可能性があります。

Machine Config Operator (MCO) は Machine Config Daemon (MCD) を使用して、設定ドリフトがないかノードを定期的に確認します。検出されると、MCO はノードおよびマシン設定プール (MCP) を **Degraded** に設定し、エラーを報告します。劣化したノードはオンラインで動作していますが、更新できません。

MCD は、以下の状況の時に設定ドリフトの検出を実行します。

- ノードがブートする時。
- マシン設定で指定されたファイル (Ignition ファイルと systemd ドロップインユニット) がマシン設定以外で変更された時。
- 新しいマシン設定が適用される前。



注記

新しいマシン設定をノードに適用すると、MCD は設定ドリフトの検出を一時的に停止します。新しいマシン設定はノード上のマシン設定とは必ず異なるため、この停止処理が必要です。新しいマシン設定が適用された後に、MCD は新しいマシン設定を使用して設定ドリフトの検出を再開します。

設定ドリフトの検出を実行する際に、MCD はファイルの内容とパーミッションが、現在適用されているマシン設定で指定されるものに完全に一致することを確認します。通常、MCD は検出がトリガーされてから 2 秒未満で設定ドリフトを検出します。

MCD が設定ドリフトを検出すると、MCD は以下のタスクを実行します。

- コンソールログにエラーを出力する
- Kubernetes イベントを生成する
- ノードでのそれ以上の検出を停止する
- ノードおよび MCP を **degraded** に設定する

MCP を一覧表示して、パフォーマンスが低下したノードがあるかどうかを確認できます。

```
$ oc get mcp worker
```

パフォーマンスが低下した MCP がある場合、以下の出力のように **DEGRADEDMACHINECOUNT** フィールドの値がゼロ以外になります。

出力例

```
NAME      CONFIG                                UPDATED  UPDATING  DEGRADED
MACHINECOUNT READYMACHINECOUNT UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT AGE
worker    rendered-worker-404caf3180818d8ac1f50c32f14b57c3  False   True     True     2
1          1          1          5h51m
```

マシン設定プールを調べることで、設定ドリフトによって問題が発生しているかどうかを判別できます。

```
$ oc describe mcp worker
```

出力例

```
...
Last Transition Time: 2021-12-20T18:54:00Z
Message:      Node ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4 is reporting: "content mismatch
for file "/etc/mco-test-file/" ❶
Reason:      1 nodes are reporting degraded status on sync
Status:      True
Type:      NodeDegraded ❷
...
```

❶ このメッセージは、マシン設定によって追加されたノードの **/etc/mco-test-file** ファイルが、マシン設定外で変更されていることを示しています。

❷ ノードの状態は **NodeDegraded** です。

あるいは、パフォーマンスが低下しているノードが分かっている場合は、そのノードを確認します。

```
$ oc describe node/ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4
```

出力例

```

...
Annotations:      cloud.network.openshift.io/egress-ipconfig: [{"interface":"nic0","ifaddr":
{"ipv4":"10.0.128.0/17"},"capacity":{"ip":10}}]
                  csi.volume.kubernetes.io/nodeid:
{"pd.csi.storage.gke.io":"projects/openshift-gce-devel-ci/zones/us-central1-
a/instances/ci-ln-j4h8nkb-72292-pxqzx-worker-a-fjks4"}
                  machine.openshift.io/machine: openshift-machine-api/ci-ln-j4h8nkb-72292-pxqzx-worker-
a-fjks4
                  machineconfiguration.openshift.io/controlPlaneTopology: HighlyAvailable
                  machineconfiguration.openshift.io/currentConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                  machineconfiguration.openshift.io/desiredConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                  machineconfiguration.openshift.io/reason: content mismatch for file "/etc/mco-test-file"
1
                  machineconfiguration.openshift.io/state: Degraded 2
...

```

1 エラーメッセージは、ノードと一覧表示されたマシン設定の間で設定ドリフトが検出されたことを示しています。このエラーメッセージは、マシン設定によって追加された **/etc/mco-test-file** の内容が、マシン設定以外で変更されていることを示しています。

2 ノードの状態は **Degraded** です。

以下の修復策のいずれかを実行して、設定ドリフトを修正し、ノードを **Ready** の状態に戻すことができます。

- ノード上のファイルの内容とパーミッションがマシン設定で設定される内容と一致するようにします。手動でファイルの内容を書き換えたり、ファイルパーミッション変更したりすることができます。
- パフォーマンスが低下したノードで **強制ファイル** を生成します。強制ファイルにより、MCD は通常の設定ドリフトの検出をバイパスし、現在のマシン設定を再度適用します。



注記

ノード上で強制ファイルを生成すると、そのノードが再起動されます。

4.1.4. マシン設定プールのステータスの確認

Machine Config Operator (MCO)、そのサブコンポーネント、およびこれが管理するリソースのステータスを表示するには、以下の **oc** コマンドを使用します。

手順

- 各マシン設定プール (MCP) のクラスターで使用可能な MCO 管理ノードの数を確認するには、次のコマンドを実行します。

```
$ oc get machineconfigpool
```

出力例

NAME	CONFIG	UPDATED	UPDATING	DEGRADED	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT	DEGRADEDMACHINECOUNT
AGE								
master	rendered-master-06c9c4...	True	False	False	3	3	3	
0	4h42m							
worker	rendered-worker-f4b64...	False	True	False	3	2	2	
0	4h42m							

ここでは、以下ようになります。

UPDATED

True ステータスは、MCO が現在のマシン設定をその MCP のノードに適用したことを示します。現在のマシン設定は、**oc get mcp** 出力の **STATUS** フィールドに指定されています。**False** ステータスは、MCP 内のノードが更新中であることを示します。

UPDATING

True ステータスは、MCO が、**MachineConfigPool** カスタムリソースで指定された目的のマシン設定を、その MCP 内の少なくとも1つのノードに適用していることを示します。目的のマシン設定は、新しく編集されたマシン設定です。更新中のノードは、スケジューリングに使用できない場合があります。**False** ステータスは、MCP 内のすべてのノードが更新されたことを示します。

DEGRADED

True ステータスは、MCO がその MCP 内の少なくとも1つのノードに現在のまたは目的のマシン設定を適用することをブロックされているか、設定が失敗していることを示します。機能が低下したノードは、スケジューリングに使用できない場合があります。**False** ステータスは、MCP 内のすべてのノードの準備ができていることを示します。

MACHINECOUNT

その MCP 内のマシンの総数を示します。

READYMACHINECOUNT

スケジューリングの準備ができているその MCP 内のマシンの総数を示します。

UPDATEDMACHINECOUNT

現在のマシン設定を持つ MCP 内のマシンの総数を示します。

DEGRADEDMACHINECOUNT

機能低下または調整不能としてマークされている、その MCP 内のマシンの総数を示します。

前の出力では、3つのコントロールプレーン(マスター)ノードと3つのワーカーノードがあります。コントロールプレーン MCP と関連するノードは、現在のマシン設定に更新されます。ワーカー MCP のノードは、目的のマシン設定に更新されています。**UPDATEDMACHINECOUNT** が 2 であることからわかるように、ワーカー MCP 内の 2 つのノードが更新され、1つがまだ更新中です。**DEGRADEDMACHINECOUNT** が 0 で、**DEGRADE** が **False** であることからわかるように、問題はありません。

MCP のノードが更新されている間、**CONFIG** の下にリストされているマシン設定は、MCP の更新元である現在のマシン設定です。更新が完了すると、リストされたマシン設定は、MCP が更新された目的のマシン設定になります。

注記

ノードが遮断されている場合、そのノードは **READYMACHINECOUNT** には含まれませんが、**MACHINECOUNT** には含まれます。また、MCP ステータスは **UPDATING** に設定されます。ノードには現在のマシン設定があるため、**UPDATEDMACHINECOUNT** の合計にカウントされます。

出力例

NAME	CONFIG	UPDATED	UPDATING	DEGRADED	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT	DEGRADEDMACHINECOUNT	AGE
master	rendered-master-06c9c4...	True	False	False	3	3	0	0	4h42m
worker	rendered-worker-c1b41a...	False	True	False	3	2	0	0	4h42m

2. **MachineConfigPool** カスタムリソースを調べて MCP 内のノードのステータスを確認するには、次のコマンドを実行します。

```
$ oc describe mcp worker
```

出力例

```
...
Degraded Machine Count: 0
Machine Count: 3
Observed Generation: 2
Ready Machine Count: 3
Unavailable Machine Count: 0
Updated Machine Count: 3
Events: <none>
```

注記

ノードが遮断されている場合、そのノードは **Ready Machine Count** に含まれません。**Unavailable Machine Count** に含まれます。

出力例

```
...
Degraded Machine Count: 0
Machine Count: 3
Observed Generation: 2
Ready Machine Count: 2
Unavailable Machine Count: 1
Updated Machine Count: 3
```

3. 既存の各 **MachineConfig** オブジェクトを表示するには、次のコマンドを実行します。

```
$ oc get machineconfigs
```

出力例

NAME	GENERATEDBYCONTROLLER	IGNITIONVERSION	AGE
00-master	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m
00-worker	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m
01-master-container-runtime	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m
01-master-kubelet	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m
...			
rendered-master-dde...	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m
rendered-worker-fde...	2c9371fbb673b97a6fe8b1c52...	3.2.0	5h18m

rendered として一覧表示された **MachineConfig** オブジェクトが変更されたり、削除されたりすることが意図されていないことに注意してください。

- 特定のマシン設定 (この場合は **01-master-kubelet**) の内容を表示するには、次のコマンドを実行します。

```
$ oc describe machineconfigs 01-master-kubelet
```

コマンドの出力は、この **MachineConfig** オブジェクトに設定ファイル (**cloud.conf** および **kubelet.conf**) と systemd サービス (Kubernetes Kubelet) の両方が含まれていることを示しています。

出力例

```
Name:      01-master-kubelet
...
Spec:
  Config:
    Ignition:
      Version: 3.2.0
    Storage:
      Files:
        Contents:
          Source: data:,
          Mode:    420
          Overwrite: true
          Path:    /etc/kubernetes/cloud.conf
        Contents:
          Source:
data:,kind%3A%20KubeletConfiguration%0AapiVersion%3A%20kubelet.config.k8s.io%2Fv1beta1%0Aauthentication%3A%0A%20%20x509%3A%0A%20%20%20%20clientCAFile%3A%20%2Fetc%2Fkubernetes%2Fkubernetes-ca.crt%0A%20%20anonymous...
          Mode:    420
          Overwrite: true
          Path:    /etc/kubernetes/kubelet.conf
      Systemd:
        Units:
          Contents: [Unit]
        Description=Kubernetes Kubelet
        Wants=rpc-statd.service network-online.target crio.service
        After=network-online.target crio.service
```

```
ExecStart=/usr/bin/hyperkube \
  kubelet \
  --config=/etc/kubernetes/kubelet.conf \ ...
```

適用するマシン設定で問題が発生した場合は、この変更を常に取り消すことができます。たとえば、**oc create -f ./myconfig.yaml** を実行してマシン設定を適用した場合、次のコマンドを実行してそのマシン設定を削除できます。

```
$ oc delete -f ./myconfig.yaml
```

これが唯一の問題である場合、影響を受けるプールのノードは動作が低下していない状態に戻るはずです。これにより、レンダリングされた設定は、直前のレンダリングされた状態にロールバックされます。

独自のマシン設定をクラスターに追加する場合、直前の例に示されたコマンドを使用して、それらのステータスと、それらが適用されるプールの関連するステータスを確認できます。

4.2. MACHINECONFIG オブジェクトを使用したノードの設定

このセクションのタスクを使用して、**MachineConfig** オブジェクトを作成し、OpenShift Container Platform ノードで実行されているファイル、systemd ユニットファイルその他のオペレーティングシステム機能を変更することができます。マシン設定の使用に関する詳細は、SSH 認証キーの [更新](#)、[イメージ署名の検証](#)、[SCTP の有効化](#)、および OpenShift Container Platform の [iSCSI イニシエーター名の設定](#) に関するコンテンツを参照してください。

OpenShift Container Platform は [Ignition 仕様バージョン 3.2](#) をサポートします。今後作成する新規のマシン設定はすべて Ignition 仕様バージョン 3.2 をベースとする必要があります。OpenShift Container Platform クラスターをアップグレードする場合、既存の Ignition 仕様バージョン 2.x マシン設定は仕様バージョン 3.2 に自動的に変換されます。

ノードの設定が、現在適用されているマシン設定で指定されているものと完全に一致しない場合があります。この状態は **設定ドリフト** と呼ばれます。Machine Config Daemon (MCD) は、ノードの設定ドリフトを定期的にチェックします。MCD が設定のドリフトを検出した場合は、管理者がノード設定を修正するまで、MCO はノードを **劣化** とマークします。劣化したノードはオンラインで動作していますが、更新できません。設定ドリフトの詳細は、[Understanding configuration drift detection](#) を参照してください。

ヒント

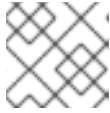
他の設定ファイルを OpenShift Container Platform ノードに追加する方法については、以下の chrony タイムサービスの設定の手順をモデルとして使用します。

4.2.1. chrony タイムサービスの設定

chrony タイムサービス (**chronyd**) で使用されるタイムサーバーおよび関連する設定は、**chrony.conf** ファイルのコンテンツを変更し、それらのコンテンツをマシン設定としてノードに渡して設定できます。

手順

1. **chrony.conf** ファイルのコンテンツを含む Butane 設定を作成します。たとえば、ワーカーノードで chrony を設定するには、**99-worker-chrony.bu** ファイルを作成します。



注記

Butane の詳細は、Butane を使用したマシン設定の作成を参照してください。

```

variant: openshift
version: 4.10.0
metadata:
  name: 99-worker-chrony ❶
  labels:
    machineconfiguration.openshift.io/role: worker ❷
storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644 ❸
      overwrite: true
      contents:
        inline: |
          pool 0.rhel.pool.ntp.org iburst ❹
          driftfile /var/lib/chrony/drift
          makestep 1.0 3
          rtsync
          logdir /var/log/chrony

```

- ❶ ❷ コントロールプレーンノードでは、これらの両方の場所で **worker** の代わりに **master** を使用します。
- ❸ マシン設定ファイルの **mode** フィールドに 8 進数の値でモードを指定します。ファイルを作成し、変更を適用すると、**mode** は 10 進数の値に変換されます。コマンド **oc get mc <mc-name> -o yaml** で YAML ファイルを確認できます。
- ❹ DHCP サーバーが提供するものなど、有効な到達可能なタイムソースを指定します。または、NTP サーバーの **1.rhel.pool.ntp.org**、**2.rhel.pool.ntp.org**、または **3.rhel.pool.ntp.org** のいずれかを指定できます。

2. Butane を使用して、ノードに配信される設定を含む **MachineConfig** オブジェクトファイル (**99-worker-chrony.yaml**) を生成します。

```
$ butane 99-worker-chrony.bu -o 99-worker-chrony.yaml
```

3. 以下の 2 つの方法のいずれかで設定を適用します。

- クラスターがまだ起動していない場合は、マニフェストファイルを生成した後、**MachineConfig** オブジェクトファイルを **<installation_directory>/openshift** ディレクトリに追加してから、クラスターの作成を続行します。
- クラスターがすでに実行中の場合は、ファイルを適用します。

```
$ oc apply -f ./99-worker-chrony.yaml
```

関連情報

- [Butane でのマシン設定の作成](#)

4.2.2. chrony タイムサービスの無効化

MachineConfig カスタムリソース (CR) を使用して、特定のロールを持つノードの chrony タイムサービス (**chronyd**) を無効にすることができます。

前提条件

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

手順

1. 指定されたノードロールの **chronyd** を無効にする **MachineConfig** CR を作成します。
  - a. 以下の YAML を **disable-chronyd.yaml** ファイルに保存します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role> ❶
  name: disable-chronyd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpd.service sntp.service ntpd.service
            Conflicts=ntpd.service systemd-timesyncd.service
            ConditionCapability=CAP_SYS_TIME
            [Service]
            Type=forking
            PIDFile=/run/chrony/chronyd.pid
            EnvironmentFile=-/etc/sysconfig/chronyd
            ExecStart=/usr/sbin/chronyd $OPTIONS
            ExecStartPost=/usr/libexec/chrony-helper update-daemon
            PrivateTmp=yes
            ProtectHome=yes
            ProtectSystem=full
            [Install]
            WantedBy=multi-user.target
          enabled: false
          name: "chronyd.service"
```

- ❶ **chronyd** を無効にするノードロール (例: **master**)。

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

```
$ oc create -f disable-chronyd.yaml
```

■

4.2.3. カーネル引数のノードへの追加

特殊なケースとして、クラスターのノードセットにカーネル引数を追加する必要がある場合があります。これは十分に注意して実行する必要があり、設定する引数による影響を十分に理解している必要があります。



警告

カーネル引数を正しく使用しないと、システムが起動不可能になる可能性があります。

設定可能なカーネル引数の例には、以下が含まれます。

- **enforcing=0**: SELinux (Security Enhanced Linux) を Permissive モードで実行するように設定します。Permissive モードでは、システムは、SELinux が読み込んだセキュリティポリシーを実行しているかのように動作します。これには、オブジェクトのラベル付けや、アクセスを拒否したエントリをログに出力するなどの動作が含まれますが、いずれの操作も拒否される訳ではありません。Permissive モードは、実稼働システムでの使用はサポートされませんが、デバッグには役に立ちます。
- **nosmt**: カーネルの対称マルチスレッド (SMT) を無効にします。マルチスレッドは、各 CPU の複数の論理スレッドを許可します。潜在的なクロススレッド攻撃に関連するリスクを減らすために、マルチテナント環境での **nosmt** の使用を検討できます。SMT を無効にすることは、基本的にパフォーマンスよりもセキュリティを重視する選択をしていることになります。
- **systemd.unified_cgroup_hierarchy**: [Linux コントロールグループバージョン 2](#) (cgroup v2) を有効にします。cgroup v2 は、カーネル [コントロールグループ](#) の次のバージョンであり、複数の改善を提供します。



重要

OpenShift Container Platform cgroups バージョン 2 機能は Developer プレビューとして提供されており、現時点では Red Hat ではサポートされていません。

カーネル引数の一覧と説明については、[Kernel.org カーネルパラメーター](#) を参照してください。

次の手順では、以下を特定する **MachineConfig** オブジェクトを作成します。

- カーネル引数を追加する一連のマシン。この場合、ワーカーロールを持つマシン。
- 既存のカーネル引数の最後に追加されるカーネル引数。
- マシン設定の一覧で変更が適用される場所を示すラベル。

前提条件

- 作業用の OpenShift Container Platform クラスターに対する管理者権限が必要です。

手順

1. OpenShift Container Platform クラスターの既存の **MachineConfig** を一覧表示し、マシン設定にラベルを付ける方法を判別します。

```
$ oc get MachineConfig
```

出力例

NAME	GENERATEDBYCONTROLLER
IGNITIONVERSION AGE	
00-master 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0
00-worker 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0
01-master-container-runtime 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
01-master-kubelet 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
01-worker-container-runtime 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
01-worker-kubelet 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
99-master-generated-registries 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
99-master-ssh 3.2.0 40m	
99-worker-generated-registries 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
99-worker-ssh 3.2.0 40m	
rendered-master-23e785de7587df95a4b517e0647e5ab7 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0 33m	
rendered-worker-5d596d9293ca3ea80c896a1191735bb1 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0 33m	

2. カーネル引数を識別する **MachineConfig** オブジェクトファイルを作成します (例: **05-worker-kernelarg-selinuxpermissive.yaml**)。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 05-worker-kernelarg-selinuxpermissive 2
spec:
  kernelArguments:
    - enforcing=0 3
```

- 1** 新しいカーネル引数をワーカーノードのみに適用します。
- 2** マシン設定 (05) 内の適切な場所を特定するための名前が指定されます (SELinux permissive モードを設定するためにカーネル引数を追加します)。
- 3** 正確なカーネル引数を **enforcing=0** として特定します。

3. 新規のマシン設定を作成します。

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. マシン設定で新規の追加内容を確認します。

```
$ oc get MachineConfig
```

出力例

NAME	GENERATEDBY	CONTROLLER
IGNITIONVERSION AGE		
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0
33m		
00-worker	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0
33m		
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
05-worker-kernelarg-selinuxpermissive		3.2.0 105s
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
99-master-ssh		3.2.0 40m
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
99-worker-ssh		3.2.0 40m
rendered-master-23e785de7587df95a4b517e0647e5ab7		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0	33m

5. ノードを確認します。

```
$ oc get nodes
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-136-161.ec2.internal	Ready	worker	28m	v1.23.0
ip-10-0-136-243.ec2.internal	Ready	master	34m	v1.23.0
ip-10-0-141-105.ec2.internal	Ready,SchedulingDisabled	worker	28m	v1.23.0
ip-10-0-142-249.ec2.internal	Ready	master	34m	v1.23.0
ip-10-0-153-11.ec2.internal	Ready	worker	28m	v1.23.0
ip-10-0-153-150.ec2.internal	Ready	master	34m	v1.23.0

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。

6. ワーカーノードのいずれかに移動し、カーネルコマンドライン引数 (ホストの `/proc/cmdline` 内) を一覧表示して、カーネル引数が機能することを確認します。

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

出力例

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

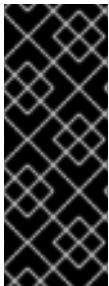
sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

enforcing=0 引数が他のカーネル引数に追加されていることを確認できるはずです。

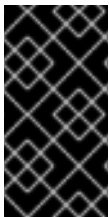
4.2.4. RHCOS のカーネル引数でのマルチパスの有効化

Red Hat Enterprise Linux CoreOS (RHCOS) はプライマリーディスクでのマルチパスをサポートするようになり、ハードウェア障害に対する対障害性が強化され、ホストの可用性を強化できるようになりました。インストール後のサポートは、マシン設定を使用してマルチパスをアクティベートすることで利用できます。



重要

インストール時のマルチパスの有効化が、OpenShift Container Platform 4.8 以降でプロビジョニングされるノードでサポートおよび推奨されるようになりました。非最適化パスに対して I/O があると、I/O システムエラーが発生するように設定するには、インストール時にマルチパスを有効にする必要があります。インストール時にマルチパスを有効にする方法は、**ベアメタルへのインストールの RHCOS** でのカーネル引数を使用したマルチパスの有効化を参照してください。



重要

IBM Z および LinuxONE では、インストール時にクラスターを設定した場合のみマルチパスを有効にできます。詳細は、**IBM Z および LinuxONE への z/VM を使用したクラスターのインストールの RHCOS** のインストールおよび OpenShift Container Platform ブートストラッププロセスの開始を参照してください。

前提条件

- バージョン 4.7 以降を使用する OpenShift Container Platform クラスターが実行中である。
- 管理者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。
- ディスクでマルチパスが有効になっていることを確認しました。マルチパスは、HBA アダプターを介して SAN に接続されているホストでのみサポートされます。

手順

1. コントロールプレーンノードでマルチパスのポストインストールを有効にするには、以下を実行します。
  - 以下の例のように、**master** ラベルを追加し、マルチパスカーネル引数を特定するようクラスターに指示する **99-master-kargs-mpath.yaml** などのマシン設定ファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "master"
  name: 99-master-kargs-mpath
spec:
  kernelArguments:
    - 'rd.multipath=default'
    - 'root=/dev/disk/by-label/dm-mpath-root'
```

2. ワーカーノードでマルチパスのポストインストールを有効にするには、以下を実行します。

- **worker** ラベルを追加し、マルチパスカーネル引数などを特定するようクラスターに指示する **99-worker-kargs-mpath.yaml** などのマシン設定ファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
  name: 99-worker-kargs-mpath
spec:
  kernelArguments:
    - 'rd.multipath=default'
    - 'root=/dev/disk/by-label/dm-mpath-root'
```

3. 以前に作成したマスターまたはワーカー YAML ファイルのいずれかを使用して新規のマシン設定を作成します。

```
$ oc create -f ./99-worker-kargs-mpath.yaml
```

4. マシン設定で新規の追加内容を確認します。

```
$ oc get MachineConfig
```

出力例

NAME	GENERATEDBYCONTROLLER
IGNITIONVERSION AGE	
00-master 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0
00-worker 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0
01-master-container-runtime 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
01-master-kubelet 3.2.0 33m	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9

```

01-worker-container-runtime          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
01-worker-kubelet                    52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
99-master-generated-registries       52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
99-master-ssh                        3.2.0      40m
99-worker-generated-registries       52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      33m
99-worker-kargs-mpath               52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0      105s
99-worker-ssh                        3.2.0      40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0      33m

```

5. ノードを確認します。

```
$ oc get nodes
```

出力例

```

NAME                                STATUS    ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal        Ready    worker   28m   v1.23.0
ip-10-0-136-243.ec2.internal        Ready    master   34m   v1.23.0
ip-10-0-141-105.ec2.internal        Ready,SchedulingDisabled worker   28m   v1.23.0
ip-10-0-142-249.ec2.internal        Ready    master   34m   v1.23.0
ip-10-0-153-11.ec2.internal         Ready    worker   28m   v1.23.0
ip-10-0-153-150.ec2.internal        Ready    master   34m   v1.23.0

```

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。

6. ワーカーノードのいずれかに移動し、カーネルコマンドライン引数 (ホストの **/proc/cmdline** 内) を一覧表示して、カーネル引数が機能することを確認します。

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

出力例

```

Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
...
rd.multipath=default root=/dev/disk/by-label/dm-mpath-root
...

sh-4.2# exit

```

追加したカーネル引数が表示されるはずです。

- インストール時のマルチパスの有効化の詳細は、[RHCOS のカーネル引数でのマルチパスの有効化](#) を参照してください。

4.3. LINUX コントロールグループバージョン 2 (CGROUPS V2) の有効化

マシン設定を使用して、クラスター内の特定ノードで [Linux コントロールグループバージョン 2](#) (cgroups v2) を有効化できます。cgroups v2 を有効にする OpenShift Container Platform プロセスにより、cgroups バージョン 1 コントローラーおよび階層がすべて無効になります。



重要

OpenShift Container Platform cgroups バージョン 2 機能は Developer プレビューとして提供されており、現時点では Red Hat ではサポートされていません。

前提条件

- バージョン 4.10 以降を使用する OpenShift Container Platform クラスターが実行中である。
- 管理者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。
- 設定するノードの **node-role.kubernetes.io** 値を把握している。

```
$ oc describe node <node-name>
```

出力例

```
Name:          ci-ln-v05w5m2-72292-5s9ht-worker-a-r6fpg
Roles:         worker
Labels:        beta.kubernetes.io/arch=amd64
               beta.kubernetes.io/instance-type=n1-standard-4
               beta.kubernetes.io/os=linux
               failure-domain.beta.kubernetes.io/region=us-central1
               failure-domain.beta.kubernetes.io/zone=us-central1-a
               kubernetes.io/arch=amd64
               kubernetes.io/hostname=ci-ln-v05w5m2-72292-5s9ht-worker-a-r6fpg
               kubernetes.io/os=linux
               node-role.kubernetes.io/worker= ❶
#...
```

❶ この値は、必要なノードロールです。

手順

- ノードで cgroups v2 を有効にします。
  - worker-cgroups-v2.yaml** などのマシン設定ファイル YAML を作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker" ❶
```



```

name: worker-enable-cgroups-v2
spec:
  kernelArguments:
    - systemd.unified_cgroup_hierarchy=1 ❷
    - cgroup_no_v1="all" ❸

```

❶ 設定するノードの **node-role.kubernetes.io** 値を指定します (例: **master**、**worker**、または **infra**)。

❷ systemd で cgroups v2 を有効にします。

❸ cgroups v1 を無効にします。

- 新規のマシン設定を作成します。

```
$ oc create -f worker-enable-cgroups-v2.yaml
```

2. マシン設定で新規の追加内容を確認します。

```
$ oc get MachineConfig
```

出力例

NAME	GENERATEDBY	CONTROLLER
IGNITIONVERSION AGE		
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0
33m		
00-worker	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.2.0
33m		
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
99-master-ssh		3.2.0 40m
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.2.0 33m		
99-worker-ssh		3.2.0 40m
rendered-master-23e785de7587df95a4b517e0647e5ab7		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0		33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.2.0		33m
worker-enable-cgroups-v2		3.2.0 10s

3. ノードを確認して、影響を受ける各ノードでスケジューリングが無効にされていることを確認します。これは、変更が適用されていることを示しています。

```
$ oc get nodes
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-fm1qjwt-72292-99kt6-master-0	Ready		master 58m	v1.23.0
ci-ln-fm1qjwt-72292-99kt6-master-1	Ready		master 58m	v1.23.0
ci-ln-fm1qjwt-72292-99kt6-master-2	Ready		master 58m	v1.23.0
ci-ln-fm1qjwt-72292-99kt6-worker-a-h5gt4	Ready,SchedulingDisabled	worker	48m	v1.23.0
ci-ln-fm1qjwt-72292-99kt6-worker-b-7vtmd	Ready	worker	48m	v1.23.0
ci-ln-fm1qjwt-72292-99kt6-worker-c-rhzhkv	Ready	worker	48m	v1.23.0

4. ノードが **Ready** 状態に戻ったら、**sys/fs/cgroup/cgroup.controllers** ファイルがノードに存在することを確認して、cgroups v2 が有効になっていることを確認できます。このファイルは cgroups v2 によって作成されます。

- そのノードのデバッグセッションを開始します。

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

- **sys/fs/cgroup/cgroup.controllers** ファイルを探します。このファイルが存在する場合は、cgroups v2 がそのノードで有効になっています。

出力例

```
cgroup.controllers cgroup.stat cpuset.cpus.effective io.stat pids
cgroup.max.depth cgroup.subtree_control cpuset.mems.effective kubepods.slice
system.slice
cgroup.max.descendants cgroup.threads init.scope memory.pressure user.slice
cgroup.procs cpu.pressure io.pressure memory.stat
```

関連情報

- インストール時に cgroups v2 を有効にする方法は、インストールプロセスの **Installation configuration parameters** セクションの **Optional parameters** の表を参照してください。

4.3.1. リアルタイムカーネルのノードへの追加

一部の OpenShift Container Platform ワークロードには、高度な決定論的アプローチが必要になります。Linux はリアルタイムのオペレーティングシステムではありませんが、Linux のリアルタイムカーネルには、リアルタイムの特性を持つオペレーティングシステムを提供するプリエンプティブなスケジューラーが含まれます。

OpenShift Container Platform ワークロードでこれらのリアルタイムの特性が必要な場合、マシンを Linux のリアルタイムカーネルに切り替えることができます。OpenShift Container Platform 4.10 の場合、**MachineConfig** オブジェクトを使用してこの切り替えを行うことができます。変更はマシン設定の **kernelType** 設定を **realtime** に変更するだけで簡単に行えますが、この変更を行う前に他のいくつかの点を考慮する必要があります。

- 現在、リアルタイムカーネルはワーカーノードでのみサポートされ、使用できるのはラジオアクセスネットワーク (RAN) のみにになります。
- 以下の手順は、Red Hat Enterprise Linux for Real Time 8 で認定されているシステムを使用したベアメタルのインストールで完全にサポートされます。

- OpenShift Container Platform でのリアルタイムサポートは、特定のサブスクリプションに制限されます。
- 以下の手順は、Google Cloud Platform での使用についてもサポートされます。

前提条件

- OpenShift Container Platform クラスター (バージョン 4.4 以降) が実行中である。
- 管理者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. リアルタイムカーネルのマシン設定を作成します。**realtime** カーネルタイプの **MachineConfig** オブジェクトが含まれる YAML ファイル (**99-worker-realtime.yaml** など) を作成します。以下の例では、すべてのワーカーノードにリアルタイムカーネルを使用するようにクラスターに指示します。

```
$ cat << EOF > 99-worker-realtime.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker"
  name: 99-worker-realtime
spec:
  kernelType: realtime
EOF
```

2. マシン設定をクラスターに追加します。以下を入力してマシン設定をクラスターに追加します。

```
$ oc create -f 99-worker-realtime.yaml
```

3. リアルタイムカーネルを確認します。影響を受けるそれぞれのノードの再起動後に、クラスターにログインして以下のコマンドを実行し、リアルタイムカーネルが設定されたノードのセットの通常のカーネルを置き換えていることを確認します。

```
$ oc get nodes
```

出力例

```
NAME                                STATUS ROLES  AGE  VERSION
ip-10-0-143-147.us-east-2.compute.internal Ready  worker  103m v1.23.0
ip-10-0-146-92.us-east-2.compute.internal Ready  worker  101m v1.23.0
ip-10-0-169-2.us-east-2.compute.internal Ready  worker  102m v1.23.0
```

```
$ oc debug node/ip-10-0-143-147.us-east-2.compute.internal
```

出力例

```
Starting pod/ip-10-0-143-147us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`
```

```
sh-4.4# uname -a
Linux <worker_node> 4.18.0-147.3.1.rt24.96.el8_1.x86_64 #1 SMP PREEMPT RT
Wed Nov 27 18:29:55 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

カーネル名には **rt** が含まれ、PREEMPT RT のテキストは、これがリアルタイムカーネルであることを示します。

4. 通常のカーネルに戻るには、**MachineConfig** オブジェクトを削除します。

```
$ oc delete -f 99-worker-realtime.yaml
```

4.3.2. journald の設定

OpenShift Container Platform ノードで **journald** サービスの設定が必要な場合は、適切な設定ファイルを変更し、そのファイルをマシン設定としてノードの適切なプールに渡すことで実行できます。

この手順では、**/etc/systemd/journald.conf** ファイルの **journald** 速度制限の設定を変更し、それらをワーカーノードに適用する方法について説明します。このファイルの使用方法についての情報は、**journald.conf** man ページを参照してください。

前提条件

- OpenShift Container Platform クラスターが実行中である。
- 管理者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. 必要な設定で **/etc/systemd/journald.conf** ファイルが含まれる Butane 設定ファイル **40-worker-custom-journald.bu** を作成します。



注記

Butane の詳細は、Butane を使用したマシン設定の作成を参照してください。

```
variant: openshift
version: 4.10.0
metadata:
  name: 40-worker-custom-journald
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/systemd/journald.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          # Disable rate limiting
          RateLimitInterval=1s
          RateLimitBurst=10000
```

```
Storage=volatile
Compress=no
MaxRetentionSec=30s
```

2. Butane を使用して、ワーカーノードに配信される設定を含む **MachineConfig** オブジェクトファイル (**40-worker-custom-journald.yaml**) を生成します。

```
$ butane 40-worker-custom-journald.bu -o 40-worker-custom-journald.yaml
```

3. マシン設定をプールに適用します。

```
$ oc apply -f 40-worker-custom-journald.yaml
```

4. 新規マシン設定が適用され、ノードの状態が低下した状態にないことを確認します。これには数分の時間がかかる場合があります。各ノードで新規マシン設定が正常に適用されるため、ワーカープールには更新が進行中であることが表示されます。

```
$ oc get machineconfigpool
NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True False False 3 3 3 0
34m
worker rendered-worker-d8 False True False 3 1 1 0
34m
```

5. 変更が適用されたことを確認するには、ワーカーノードにログインします。

```
$ oc get node | grep worker
ip-10-0-0-1.us-east-2.compute.internal Ready worker 39m v0.0.0-master+$Format:%h$
$ oc debug node/ip-10-0-0-1.us-east-2.compute.internal
Starting pod/ip-10-0-141-142us-east-2computeinternal-debug ...
...
sh-4.2# chroot /host
sh-4.4# cat /etc/systemd/journald.conf
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
sh-4.4# exit
```

関連情報

- [Butane でのマシン設定の作成](#)

4.3.3. 拡張機能の RHCOS への追加

RHCOS はコンテナ指向の最小限の RHEL オペレーティングシステムであり、すべてのプラットフォームで OpenShift Container Platform クラスターに共通の機能セットを提供するように設計されています。ソフトウェアパッケージを RHCOS システムに追加することは一般的に推奨されていませんが、MCO は RHCOS ノードに最小限の機能セットを追加するために使用できる **extensions** 機能を提供します。

現時点で、以下の拡張機能が利用可能です。

- **usbguard**: **usbguard** 拡張機能を追加すると、RHCOS システムを割り込みの USB デバイスから保護します。詳細は、[USBGuard](#) を参照してください。

以下の手順では、マシン設定を使用して1つ以上の拡張機能を RHCOS ノードに追加する方法を説明します。

前提条件

- OpenShift Container Platform クラスター (バージョン 4.6 以降) が実行中である。
- 管理者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. 拡張機能のマシン設定を作成します。**MachineConfig extensions** オブジェクトが含まれる YAML ファイル (例: **80-extensions.yaml**) を作成します。この例では、クラスターに対して **usbguard** 拡張機能を追加するように指示します。

```
$ cat << EOF > 80-extensions.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 80-worker-extensions
spec:
  config:
    ignition:
      version: 3.2.0
    extensions:
      - usbguard
EOF
```

2. マシン設定をクラスターに追加します。以下を入力してマシン設定をクラスターに追加します。

```
$ oc create -f 80-extensions.yaml
```

これにより、すべてのワーカーノードで **usbguard** の rpm パッケージがインストールされるように設定できます。

3. 拡張機能が適用されていることを確認します。

```
$ oc get machineconfig 80-worker-extensions
```

出力例

```
NAME                                GENERATEDBYCONTROLLER IGNITIONVERSION AGE
80-worker-extensions                3.2.0                    57s
```

4. 新規マシン設定が適用され、ノードの状態が低下した状態にないことを確認します。これには数分の時間がかかる場合があります。各マシンで新規マシン設定が正常に適用されるため、ワーカープールには更新が進行中であることが表示されます。

```
$ oc get machineconfigpool
```

出力例

```
NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True False False 3 3 3 0
34m
worker rendered-worker-d8 False True False 3 1 1 0
34m
```

5. 拡張機能を確認します。拡張機能が適用されたことを確認するには、以下を実行します。

```
$ oc get node | grep worker
```

出力例

```
NAME STATUS ROLES AGE VERSION
ip-10-0-169-2.us-east-2.compute.internal Ready worker 102m v1.23.0
```

```
$ oc debug node/ip-10-0-169-2.us-east-2.compute.internal
```

出力例

```
...
To use host binaries, run `chroot /host`
sh-4.4# chroot /host
sh-4.4# rpm -q usbguard
usbguard-0.7.4-4.el8.x86_64.rpm
```

4.3.4. マシン設定マニフェストでのカスタムファームウェアブロープの読み込み

/usr/lib 内のファームウェアブロープのデフォルトの場所は読み取り専用であるため、検索パスを更新して、カスタムファームウェアブロープを特定できます。これにより、ブロープが RHCOS によって管理されない場合に、マシン設定マニフェストでローカルファームウェアブロープを読み込むことができます。

手順

1. Butane 設定ファイル **98-worker-firmware-blob.bu** を作成します。このファイルは、root 所有でローカルストレージに書き込みできるように、検索パスを更新します。以下の例では、カスタムブロープファイルをローカルワークステーションからノードの **/var/lib/firmware** 下に配置しています。



注記

Butane の詳細は、Butane を使用したマシン設定の作成を参照してください。

カスタムファームウェアプロブ用の Butane 設定ファイル

```
variant: openshift
version: 4.10.0
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 98-worker-firmware-blob
storage:
  files:
    - path: /var/lib/firmware/<package_name> ❶
      contents:
        local: <package_name> ❷
        mode: 0644 ❸
openshift:
  kernel_arguments:
    - 'firmware_class.path=/var/lib/firmware' ❹
```

- ❶ ファームウェアパッケージのコピー先となるノードのパスを設定します。
- ❷ Butane を実行しているシステムのローカルファイルディレクトリーから読み取るコンテンツを含むファイルを指定します。ローカルファイルのパスは **files-dir** ディレクトリーからの相対パスで、以下の手順の Butane で **--files-dir** オプションを使用して指定する必要があります。
- ❸ RHCOS ノードのファイルのパーミッションを設定します。**0644** パーミッションを設定することが推奨されます。
- ❹ **firmware_class.path** パラメーターは、ローカルワークステーションからノードのルートファイルシステムにコピーされたカスタムファームウェアプロブを検索するカーネルの検索パスをカスタマイズします。この例では、**/var/lib/firmware** をカスタマイズされたパスとして使用します。

2. Butane を実行して、ローカルワークステーション上の **98-worker-firmware-blob.yaml** という名前のファームウェアプロブのコピーを使用する **MachineConfig** オブジェクトファイルを作成します。ファームウェアプロブには、ノードに配信される設定が含まれます。次の例では、**--files-dir** オプションを使用して、ローカルファイルが配置されるワークステーション上のディレクトリーを指定します。

```
$ butane 98-worker-firmware-blob.bu -o 98-worker-firmware-blob.yaml --files-dir
<directory_including_package_name>
```

3. 以下の 2 つの方法のいずれかで、設定をノードに適用します。
  - クラスターがまだ起動していない場合は、マニフェストファイルを作成した後、**MachineConfig** オブジェクトファイルを **<installation_directory>/openshift** ディレクトリーに追加してから、クラスターの作成を続行します。
  - クラスターがすでに実行中の場合は、ファイルを適用します。

```
$ oc apply -f 98-worker-firmware-blob.yaml
```

MachineConfig オブジェクト YAML ファイルは、マシンの設定を終了するために作成されます。

4. 将来的に **MachineConfig** オブジェクトを更新する必要がある場合に備えて、Butane 設定を保存します。

関連情報

- [Butane でのマシン設定の作成](#)

4.4. MCO 関連のカスタムリソースの設定

MCO は **MachineConfig** オブジェクトを管理する以外にも、2つのカスタムリソース (CR) (**KubeletConfig** および **ContainerRuntimeConfig**) を管理します。これらの CR を使用すると、Kubelet および CRI-O コンテナランタイムサービスの動作に影響を与えるノードレベルの設定を変更することができます。

4.4.1. 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) を使用して kubelet パラメーターを編集できます。



注記

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

以下のガイダンスを参照してください。

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

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



注記

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

KubeletConfig CR の例

```
$ oc get kubeletconfig
```

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

KubeletConfig マシン設定を示す例

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

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

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

前提条件

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

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

以下に例を示します。

```
$ oc describe machineconfigpool worker
```

出力例

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

① ラベルが追加されると、**labels** の下に表示されます。

- b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

手順

1. これは、選択可能なマシン設定オブジェクトを表示します。

```
$ oc get machineconfig
```

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

2. ノードあたりの最大 Pod の現在の値を確認します。

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

以下に例を示します。

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

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

出力例

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

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

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

❶ Machine Config Pool からラベルを入力します。

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



注記

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

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

- a. ラベルを使用してワーカーのマシン設定プールを更新します。

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

- b. **KubeletConfig** オブジェクトを作成します。

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

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

```
$ oc get kubeletconfig
```

出力例

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

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

4. 変更がノードに適用されていることを確認します。

- a. **maxPods** 値が変更されたワーカーノードで確認します。

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

- b. **Allocatable** スタンザを見つけます。

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

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

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

5. KubeletConfig オブジェクトの変更を確認します。

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

これは、以下の例のように **True** および **type:Success** のステータスを表示します。

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

4.4.2. CRI-O パラメーターを編集するための ContainerRuntimeConfig CR の作成

特定のマシン設定プール (MCP) に関連付けられたノードの OpenShift Container Platform CRI-O ランタイムに関連付けられる設定の一部を変更することができます。**ContainerRuntimeConfig** カスタムリソース (CR) を使用して、設定値を設定し、MCP に一致するラベルを追加します。次に、MCO は関連付けられたノードで **crio.conf** および **storage.conf** 設定ファイルを更新された値を使用して再ビルドします。



注記

ContainerRuntimeConfig CR を使用して実装された変更を元に戻すには、CR を削除する必要があります。マシン設定プールからラベルを削除しても、変更は元に戻されません。

ContainerRuntimeConfig CR を使用して以下の設定を変更することができます。

- **PIDs limit:** **pidsLimit** パラメーターは、コンテナで許可されるプロセスの最大数である CRI-O **pids_limit** パラメーターを設定します。デフォルトは 1024 (**pids_limit = 1024**) です。
- **Log level:** **logLevel** パラメーターは CRI-O **log_level** パラメーターを設定します。これはログメッセージの詳細レベルです。デフォルトは **info** (**log_level = info**) です。他のオプションには、**fatal**、**panic**、**error**、**warn**、**debug**、および **trace** が含まれます。

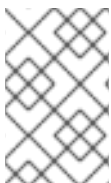
- **Overlay size:** `overlaySize` パラメーターは、コンテナイメージの最大サイズである CRI-O Overlay ストレージドライバーの `size` パラメーターを設定します。
- **Maximum log size:** `logSizeMax` パラメーターは CRI-O `log_size_max` パラメーターを設定します。これは、コンテナログファイルに許可される最大サイズです。デフォルトは無制限 (`log_size_max = -1`) です。これが正数に設定される場合、8192 以上にすることが必要があり、ConMon の読み取りバッファよりも小さくすることはできません。ConMon は、単一コンテナのコンテナマネージャー (Podman、CRI-O など) と OCI ランタイム (runc または crun など) との間の通信を監視するプログラムです。

マシン設定プールごとに、そのプールに加える設定変更をすべて含めて、**ContainerRuntimeConfig** CR を 1 つ割り当てる必要があります。同じコンテンツをすべてのプールに適用している場合には、すべてのプールに必要なのは **ContainerRuntimeConfig** CR 1 つだけです。

既存の **ContainerRuntimeConfig** CR を編集して既存の設定を編集するか、変更ごとに新規 CR を作成する代わりに新規の設定を追加する必要があります。異なるマシン設定プールを変更する場合や、変更が一時的で元に戻すことができる場合のみ、新しい **ContainerRuntimeConfig** CR の作成を推奨しています。

必要に応じて複数の **ContainerRuntimeConfig** CR を作成できます。この場合、制限はクラスターごとに 10 個となっています。最初の **ContainerRuntimeConfig** CR について、MCO は **containerruntime** で追加されたマシン設定を作成します。それぞれの後続の CR で、コントローラーは数字の接尾辞が付いた新規の **containerruntime** マシン設定を作成します。たとえば、**containerruntime** マシン設定に **-2** 接尾辞がある場合、次の **containerruntime** マシン設定が **-3** を付けて追加されます。

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



注記

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

複数の ContainerRuntimeConfig CR を示す例

```
$ oc get ctrcfg
```

出力例

```
NAME      AGE
ctr-pid   24m
ctr-overlay 15m
ctr-level 5m45s
```

複数の containerruntime マシン設定を示す例

```
$ oc get mc | grep container
```

出力例

```
...
```

```

01-master-container-runtime      b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.2.0
57m
...
01-worker-container-runtime      b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.2.0
57m
...
99-worker-generated-containerruntime  b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0      26m
99-worker-generated-containerruntime-1  b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0      17m
99-worker-generated-containerruntime-2  b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0      7m26s
...

```

以下の例では、**pids_limit** を 2048 に引き上げ、**log_level** を **debug** に設定し、オーバーレイサイズを 8 GB に設定し、**log_size_max** を無制限に設定します。

ContainerRuntimeConfig CR の例

```

apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: overlay-size
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "1"
  containerRuntimeConfig:
    pidsLimit: 2048
    logLevel: debug
    overlaySize: 8G
    logSizeMax: "-1"

```

- ❶ マシン設定プールのラベルを指定します。
- ❷ オプション: コンテナで許可されるプロセスの最大数を指定します。
- ❸ オプション: ログメッセージの詳細レベルを指定します。
- ❹ オプション: コンテナイメージの最大サイズを指定します。
- ❺ オプション: コンテナログファイルに許可される最大サイズを指定します。正の数に設定する場合は、8192 以上でなければなりません。

手順

ContainerRuntimeConfig CR を使用して CRI-O 設定を変更するには、以下を実行します。

1. **ContainerRuntimeConfig** CR の YAML ファイルを作成します。

```

apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: overlay-size

```

```
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "1"
  containerRuntimeConfig: 2
    pidsLimit: 2048
    logLevel: debug
    overlaySize: 8G
    logSizeMax: "-1"
```

- 1 変更する必要があるマシン設定プールのラベルを指定します。
- 2 必要に応じてパラメーターを設定します。

2. **ContainerRuntimeConfig** CR を作成します。

```
$ oc create -f <file_name>.yaml
```

3. CR が作成されたことを確認します。

```
$ oc get ContainerRuntimeConfig
```

出力例

```
NAME      AGE
overlay-size 3m19s
```

4. 新規の **containerruntime** マシン設定が作成されていることを確認します。

```
$ oc get machineconfigs | grep containerrun
```

出力例

```
99-worker-generated-containerruntime 2c9371fbb673b97a6fe8b1c52691999ed3a1bfc2
3.2.0 31s
```

5. すべての準備状態にあるものとして表示されるまでマシン設定プールをモニターします。

```
$ oc get mcp worker
```

出力例

```
NAME  CONFIG          UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-169 False   True     False    3          1          1          0
9h
```

6. 設定が CRI-O で適用されたことを確認します。

- a. マシン設定プールのノードに対して **oc debug** セッションを開き、**chroot /host** を実行します。


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

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

- b. **crio.conf** ファイルの変更を確認します。

```
sh-4.4# crio config | egrep 'log_level|pids_limit|log_size_max'
```

出力例

```
pids_limit = 2048
log_size_max = -1
log_level = "debug"
```

- c. 'storage.conf' ファイルの変更を確認します。

```
sh-4.4# head -n 7 /etc/containers/storage.conf
```

出力例

```
[storage]
driver = "overlay"
runroot = "/var/run/containers/storage"
graphroot = "/var/lib/containers/storage"
[storage.options]
additionalimagestores = []
size = "8G"
```

4.4.3. CRI-O を使用した Overlay のデフォルトのコンテナルートパーティションの最大サイズの設定

各コンテナのルートパーティションには、基礎となるホストの利用可能なディスク領域がすべて表示されます。以下のガイダンスに従って、すべてのコンテナのルートディスクの最大サイズを設定します。

Overlay の最大サイズ、およびログレベルや PID 制限などの他の CRI-O オプションを設定するには、以下の **ContainerRuntimeConfig** カスタムリソース定義 (CRD) を作成できます。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: overlay-size
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-crio: overlay-size
  containerRuntimeConfig:
    pidsLimit: 2048
    logLevel: debug
    overlaySize: 8G
```

手順

1. 設定オブジェクトを作成します。

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

2. 新規の CRI-O 設定をワーカーノードに適用するには、ワーカーのマシン設定プールを編集します。

```
$ oc edit machineconfigpool worker
```

3. **ContainerRuntimeConfig** CRD に設定した **matchLabels** 名に基づいて **custom-crio** ラベルを追加します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2020-07-09T15:46:34Z"
  generation: 3
  labels:
    custom-crio: overlay-size
    machineconfiguration.openshift.io/mco-built-in: ""
```

4. 変更を保存して、マシン設定を表示します。

```
$ oc get machineconfigs
```

新規の **99-worker-generated-containerruntime** および **rendered-worker-xyz** オブジェクトが作成されます。

出力例

```
99-worker-generated-containerruntime 4173030d89bf4a7a0976d1665491a4d9a6e54f1
3.2.0 7m42s
rendered-worker-xyz 4173030d89bf4a7a0976d1665491a4d9a6e54f1 3.2.0
7m36s
```

5. これらのオブジェクトの作成後に、変更が適用されるようにマシン設定プールを監視します。

```
$ oc get mcp worker
```

ワーカーノードには、マシン数、更新数およびその他の詳細と共に **UPDATING** が **True** として表示されます。

出力例

```
NAME CONFIG UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz False True False 3 2 2 0
20h
```

完了すると、ワーカーノードは **UPDATING** を **False** に戻し、**UPDATEDMACHINECOUNT** 数は **MACHINECOUNT** に一致します。

出力例

```
NAME CONFIG          UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz True    False    False    3      3      3      0
20h
```

ワーカーマシンを見ると、新規の 8 GB の最大サイズの設定がすべてのワーカーに適用されていることを確認できます。

出力例

```
head -n 7 /etc/containers/storage.conf
[storage]
driver = "overlay"
runroot = "/var/run/containers/storage"
graphroot = "/var/lib/containers/storage"
[storage.options]
additionalimagestores = []
size = "8G"
```

コンテナ内では、ルートパーティションが 8 GB であることを確認できます。

出力例

```
~ $ df -h
Filesystem      Size    Used Available Use% Mounted on
overlay         8.0G    8.0K    8.0G    0% /
```

第5章 インストール後のクラスタタスク

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

5.1. 利用可能なクラスタのカスタマイズ

OpenShift Container Platform クラスタのデプロイ後は、大半のクラスタ設定およびカスタマイズが終了していることになります。数多くの設定リソースが利用可能です。



注記

クラスタを IBM Z にインストールする場合は、すべての特長および機能が利用可能である訳ではありません。

イメージレジストリー、ネットワーク設定、イメージビルドの動作およびアイデンティティプロバイダーなどのクラスタの主要な機能を設定するために設定リソースを変更します。

これらのリソースを使用して制御する設定の現在の記述については、**oc explain** コマンドを使用します (例: **oc explain builds --api-version=config.openshift.io/v1**)。

5.1.1. クラスタ設定リソース

すべてのクラスタ設定リソースはグローバルにスコープが設定され (namespace は設定されない)、**cluster** という名前が付けられます。

リソース名	説明
apiserver.config.openshift.io	証明書および認証局 などの API サーバー設定を提供します。
authentication.config.openshift.io	クラスタの アイデンティティプロバイダー および認証設定を制御します。
build.config.openshift.io	クラスタのすべてのビルドについてのデフォルトおよび有効にされている 設定 を制御します。
console.config.openshift.io	ログアウト動作 を含む Web コンソールインターフェイスの動作を設定します。
featuregate.config.openshift.io	FeatureGates を有効にして、テクノロジープレビュー機能を使用できるようにします。
image.config.openshift.io	特定の イメージレジストリー が処理される方法を設定します (allowed、disallowed、insecure、CA details)。
ingress.config.openshift.io	ルートのデフォルトドメインなどの ルーティング に関連する設定の詳細。

リソース名	説明
oauth.config.openshift.io	内部 OAuth サーバー フローに関連するアイデンティティプロバイダーおよび他の動作を設定します。
project.config.openshift.io	プロジェクトテンプレートを含む、 プロジェクトの作成方法 を設定します。
proxy.config.openshift.io	外部ネットワークアクセスを必要とするコンポーネントで使用するプロキシを定義します。注: すべてのコンポーネントがこの値を使用する訳ではありません。
scheduler.config.openshift.io	プロファイルやデフォルトのノードセクターなどの スケジューラー の動作を設定します。

5.1.2. Operator 設定リソース

これらの設定リソースは、**cluster** という名前のクラスタースコープのインスタンスです。これは、特定の Operator によって所有される特定コンポーネントの動作を制御します。

リソース名	説明
consoles.operator.openshift.io	ブランドのカスタマイズなどのコンソールの外観の制御
config.imageregistry.operator.openshift.io	パブリックルーティング、プロキシ設定、リソース設定、レプリカ数およびストレージタイプなどの OpenShift イメージレジストリー設定 を設定します。
config.samples.operator.openshift.io	Samples Operator を設定して、クラスターにインストールされるイメージストリームとテンプレートのサンプルを制御します。

5.1.3. 追加の設定リソース

これらの設定リソースは、特定コンポーネントの単一インスタンスを表します。場合によっては、リソースの複数のインスタンスを作成して、複数のインスタンスを要求できます。他の場合には、Operator は特定の namespace の特定のリソースインスタンス名のみを使用できます。追加のリソースインスタンスの作成方法や作成するタイミングについての詳細は、コンポーネント固有のドキュメントを参照してください。

リソース名	インスタンス名	Namespace	説明
alertmanager.monitoring.coreos.com	main	openshift-monitoring	Alertmanager デプロイメントパラメーターを制御します。

リソース名	インスタンス名	Namespace	説明
<code>ingresscontroller.operator.openshift.io</code>	<code>default</code>	<code>openshift-ingress-operator</code>	ドメイン、レプリカ数、証明書、およびコントローラーの配置などの Ingress Operator 動作を設定します。

5.1.4. 情報リソース

これらのリソースを使用して、クラスターについての情報を取得します。設定によっては、これらのリソースの直接編集が必要になる場合があります。

リソース名	インスタンス名	説明
<code>clusterversion.config.openshift.io</code>	<code>version</code>	OpenShift Container Platform 4.10 では、実稼働クラスターの ClusterVersion リソースをカスタマイズすることはできません。その代わりとして、 クラスターを更新する ためのプロセスを実行します。
<code>dns.config.openshift.io</code>	<code>cluster</code>	クラスターの DNS 設定を変更することはできません。 DNS Operator ステータスを表示 できます。
<code>infrastructure.config.openshift.io</code>	<code>cluster</code>	クラスターはそのクラウドプロバイダーとの対話を可能にする設定の詳細。
<code>network.config.openshift.io</code>	<code>cluster</code>	インストール後にクラスターのネットワークを変更することはできません。ネットワークをカスタマイズするには、 インストール時にネットワークをカスタマイズするプロセス を実行します。

5.2. グローバルクラスターのプルシークレットの更新

現在のプルシークレットを置き換えるか、新しいプルシークレットを追加することで、クラスターのグローバルプルシークレットを更新できます。

ユーザーがインストール中に使用したレジストリーとは別のレジストリーを使用してイメージを保存する場合は、この手順が必要です。

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. オプション: 既存のプルシークレットに新しいプルシークレットを追加するには、以下の手順を実行します。
  - a. 以下のコマンドを入力してプルシークレットをダウンロードします。

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> ❶
```

- ❶ プルシークレットファイルへのパスを指定します。

b. 以下のコマンドを実行して、新しいプルシークレットを追加します。

```
$ oc registry login --registry="<registry>" ❶
--auth-basic="<username>:<password>" ❷
--to=<pull_secret_location> ❸
```

- ❶ 新しいレジストリーを指定します。同じレジストリー内に複数のリポジトリを含めることができます (例: `--registry="<registry/my-namespace/my-repository>"`)。
- ❷ 新しいレジストリーの認証情報を指定します。
- ❸ プルシークレットファイルへのパスを指定します。

または、プルシークレットファイルを手動で更新することもできます。

2. 以下のコマンドを実行して、クラスターのグローバルプルシークレットを更新します。

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> ❶
```

- ❶ 新規プルシークレットファイルへのパスを指定します。

この更新はすべてのノードにロールアウトされます。これには、クラスターのサイズに応じて多少時間がかかる場合があります。



注記

OpenShift Container Platform 4.7.4 の時点で、グローバルプルシークレットへの変更によってノードドレインまたは再起動がトリガーされなくなりました。

5.3. 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。

5.3.1. 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。

5.3.2. 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. クラスター内にあるマシンを表示します。

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

3. 削除するマシンに注釈を設定します。

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/cluster-api-delete-machine="true"
```

4. 次のいずれかのコマンドを実行して、コンピュート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。

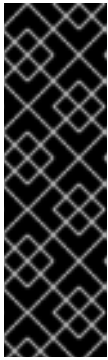
```
$ oc edit machineset <machineset> -n openshift-machine-api
```

ヒント

または、以下の YAML を適用してマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```


コンピュータ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。



重要

デフォルトでは、マシンコントローラーは、成功するまでマシンによってサポートされるノードをドレイン (解放) しようとします。Pod 中断バジェットの設定が間違っているなど、状況によっては、ドレイン操作が成功しない可能性があります。排水操作が失敗した場合、マシンコントローラーはマシンの取り外しを続行できません。

特定のマシンの **machine.openshift.io/exclude-node-draining** にアノテーションを付けると、ノードのドレイン (解放) を省略できます。

検証

- 目的のマシンの削除を確認します。

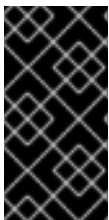
```
$ oc get machines
```

5.3.3. 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、削除ポリシーに関係なく、関連するマシンにアノテーション **machine.openshift.io/cluster-api-delete-machine=true** を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。

5.3.4. 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

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ー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。

クラスター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 をスケジューリングします。

スケジューラー Operator カスタムリソース (CR) を編集して、クラスター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ー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 1
  mastersSchedulable: false
```

- 1 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、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 1
```

- 1 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
```

ヒント

あるいは、以下の YAML を適用してマシンセットにラベルを追加することもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

- 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 abc612-msrtw-worker-us-east-1c -n openshift-machine-api
```

MachineSet オブジェクトの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
...
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  v1.23.0
```

- ラベルをノードに直接追加します。
  - 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
```

ヒント

あるいは、以下の YAML を適用してノードにラベルを追加することもできます。

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
labels:
  type: "user-node"
  region: "east"
```

- 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	v1.23.0

5.4. 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

マシンセットを作成して、デフォルトのルーター、統合コンテナイメージレジストリー、およびクラスターメトリクスおよびモニタリングのコンポーネントなどのインフラストラクチャーコンポーネントのみをホストするマシンを作成できます。これらのインフラストラクチャーマシンは、環境の実行に必要なサブスクリプションの合計数にカウントされません。

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することが推奨されます。OpenShift Logging と Red Hat OpenShift Service Mesh の両方が Elasticsearch をデプロイします。これには、3つのインスタンスを異なるノードにインストールする必要があります。これらの各ノードは、高可用性のために異なるアベイラビリティゾーンにデプロイできます。このような設定では、各アベイラビリティゾーンに1つずつ、3つの異なるマシンセットが必要です。複数のアベイラビリティゾーンを持たないグローバル Azure リージョンでは、アベイラビリティセットを使用して高可用性を確保できます。

インフラストラクチャーノードおよびインフラストラクチャーノードで実行できるコンポーネントの情報は、[Creating infrastructure machine sets](#) を参照してください。

インフラストラクチャーノードを作成するには、[マシンセット](#)、[post_installation_configuration/cluster-tasks.adoc#creating-an-infra-node_post-install-cluster-tasks\[assign a label to the nodes\]](#)、または [マシン設定プール](#) を使用できます。

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。

特定のノードセクターをすべてのインフラストラクチャーコンポーネントに適用すると、OpenShift Container Platform は [そのラベルを持つノードでそれらのワークロードをスケジュール](#) します。

5.4.1. 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コンピュートセットセットに加えて、独自のマシンセットを作成して、選択した特定のワークロードのマシンコンピューティングリソースを動的に管理できます。

前提条件

- OpenShift Container Platform クラスターをデプロイすること。
- OpenShift CLI (**oc**) がインストールされている。
- **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。
2. オプション: 特定のフィールドに設定する値がわからない場合は、クラスターから既存のコンピュートマシンセットを確認できます。

- a. クラスター内のコンピュータマシンセットをリスト表示するには、次のコマンドを実行します。

```
$ 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. 特定のコンピュータマシンセットカスタムリソース(CR)の値を表示するには、以下のコマンドを実行します。

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

出力例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> ❶
  name: <infrastructure_id>-<role> ❷
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      providerSpec: ❸
      ...
```

❶ クラスターインフラストラクチャー ID。

❷ デフォルトのノードラベル。

**注記**

ユーザーがプロビジョニングしたインフラストラクチャーを持つクラスターの場合、コンピュータマシンセットは **worker** および **infra** タイプのマシンのみを作成できます。

3

コンピュータマシンセット CR の **<providerSpec>** セクションの値は、プラットフォーム固有です。CR の **<providerSpec>** パラメーターの詳細については、プロバイダーのサンプルコンピュータマシンセット CR 設定を参照してください。

- 次のコマンドを実行して **MachineSet** CR を作成します。

```
$ oc create -f <file_name>.yaml
```

検証

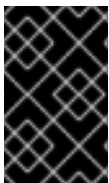
- 次のコマンドを実行して、コンピュータマシンセットのリストを表示します。

```
$ 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。

5.4.2. 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コントロールプレーンノードがマシン 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、コントロールプレーン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

- 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. デフォルトのクラスタースコープのセクターを作成するには、以下を実行します。デフォルトのノードセクターはすべての namespace で作成された Pod に適用されます。これにより、Pod の既存のノードセクターとの交差が作成され、Pod のセクターをさらに制限します。

重要

デフォルトのノードセクターのキーが Pod のラベルのキーと競合する場合、デフォルトのノードセクターは適用されません。

ただし、Pod がスケジュール対象外になる可能性のあるデフォルトノードセクターを設定しないでください。たとえば、Pod のラベルが **node-role.kubernetes.io/master=""** などの別のノードロールに設定されている場合、デフォルトのノードセクターを **node-role.kubernetes.io/infra=""** などの特定のノードロールに設定すると、Pod がスケジュール不能になる可能性があります。このため、デフォルトのノードセクターを特定のノードロールに設定する際には注意が必要です。

または、プロジェクトノードセクターを使用して、クラスター全体でのノードセクターの競合を避けることができます。

- a. **Scheduler** オブジェクトを編集します。

```
$ oc edit scheduler cluster
```

- b. 適切なノードセクターと共に **defaultNodeSelector** フィールドを追加します。

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec:
  defaultNodeSelector: topology.kubernetes.io/region=us-east-1 1
# ...
```

- 1** このサンプルノードセクターは、デフォルトで **us-east-1** リージョンのノードに Pod をデプロイします。

- c. 変更を適用するためにファイルを保存します。

これで、インフラストラクチャーリソースを新しくラベル付けされた **infra** ノードに移動できます。

関連情報

- プロジェクトノードセクターを設定してクラスター全体のノードセクターキーの競合を回避する方法に関する詳細は、[Project node selectors](#) を参照してください。

5.4.3. インフラストラクチャマシンのマシン設定プール作成

インフラストラクチャ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。

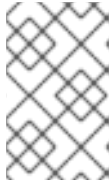
```
$ oc get machineconfig
```

出力例

NAME	GENERATEDBY	CONTROLLER
IGNITIONVERSION	CREATED	
00-master	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.2.0	31d	
00-worker	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.2.0	31d	
01-master-container-runtime	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 31d
01-master-kubelet	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.2.0	31d	
01-worker-container-runtime	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 31d
01-worker-kubelet	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.2.0	31d	
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 31d
99-master-ssh		3.2.0 31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 31d
99-worker-ssh		3.2.0 31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 23m
rendered-master-072d4b2da7f88162636902b074e9e28e	5b6fb8349a29735e48446d435962dec4547d3090	3.2.0 31d
rendered-master-3e88ec72aed3886dec061df60d16d1af	02c07496ba0417b3e12b78fb32baf6293d314f79	3.2.0 31d
rendered-master-419bee7de96134963a15fdf9dd473b25	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 17d
rendered-master-53f5c91c7661708adce18739cc0f40fb	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 13d
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 7d3h
rendered-master-dc7f874ec77fc4b969674204332da037	5b6fb8349a29735e48446d435962dec4547d3090	3.2.0 31d
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d	5b6fb8349a29735e48446d435962dec4547d3090	3.2.0 31d
rendered-worker-2640531be11ba43c61d72e82dc634ce6	5b6fb8349a29735e48446d435962dec4547d3090	3.2.0 31d
rendered-worker-4e48906dca84ee702959c71a53ee80e7	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 7d3h
rendered-worker-4f110718fe88e5f349987854a1147755	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 17d
rendered-worker-afc758e194d6188677eb837842d3b379	02c07496ba0417b3e12b78fb32baf6293d314f79	3.2.0 31d
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 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。

5. 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。

- a. 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:
  name: 51-infra
labels:
  machineconfiguration.openshift.io/role: infra ❶
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - path: /etc/infratest
        mode: 0644
        contents:
          source: data:,infra
```

- 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。

5.5. 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、ティントを、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。

5.5.1. 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、ティントおよび容認 (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

- ティントを **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。
  - ノードにティ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>=<value>:<effect>
```

以下に例を示します。

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

ヒント

または、以下の YAML を適用してテイントを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      effect: NoExecute
      value: reserved
    ...
```

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 のみをスケジューリングしますが、既存の Pod はノードにスケジューリング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リングする必要のある 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: NoExecute 1
```

```
key: node-role.kubernetes.io/infra ❷
operator: Exists ❸
value: reserved ❹
```

- 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。
- ❹ ノードに追加したキーと値のペア Taint の値を指定します。

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。

5.6. 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。

5.6.1. 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
```

```

spec:
  nodePlacement:
    nodeSelector: ❶
    matchLabels:
      node-role.kubernetes.io/infra: ""
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved

```

- ❶ 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。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

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	v1.23.0

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

5.6.2. 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
```

```

spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            namespaces:
              - openshift-image-registry
            topologyKey: kubernetes.io/hostname
            weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector: ❶
    node-role.kubernetes.io/infra: ""
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved

```

- ❶ 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。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

3. 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。

5.6.3. モニタリングソリューションの移動

監視スタックには、Prometheus、Grafana、Alertmanager などの複数のコンポーネントが含まれています。Cluster Monitoring Operator は、このスタックを管理します。モニタリングスタックをインフラストラクチャーノードに再デプロイするために、カスタム config map を作成して適用できます。

手順

1. **cluster-monitoring-config** 設定マップを編集し、**nodeSelector** を変更して **infra** ラベルを使用します。

```
$ oc edit configmap cluster-monitoring-config -n openshift-monitoring
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector: ❶
        node-role.kubernetes.io/infra: ""
      tolerations:
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoSchedule
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoExecute
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoSchedule
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoExecute
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoSchedule
        - key: node-role.kubernetes.io/infra
          value: reserved
          effect: NoExecute
    grafana:
```

```
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
k8sPrometheusAdapter:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
kubeStateMetrics:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
telemetryClient:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
openshiftStateMetrics:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
thanosQuerier:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
```

```
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoExecute
```

- 1 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。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

2. 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'
```

3. 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。

5.6.4. OpenShift Logging リソースの移動

Elasticsearch および Kibana などのロギングシステムコンポーネントの 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 をインストールし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

- Red Hat OpenShift Logging および Elasticsearch Operators がインストールされている必要があります。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
  logStore:
    elasticsearch:
```

```

nodeCount: 3
nodeSelector: ❶
  node-role.kubernetes.io/infra: "
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/infra
  value: reserved
- effect: NoExecute
  key: node-role.kubernetes.io/infra
  value: reserved
redundancyPolicy: SingleRedundancy
resources:
  limits:
    cpu: 500m
    memory: 16Gi
  requests:
    cpu: 500m
    memory: 16Gi
  storage: {}
type: elasticsearch
managementState: Managed
visualization:
  kibana:
    nodeSelector: ❷
      node-role.kubernetes.io/infra: "
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
    proxy:
      resources: null
    replicas: 1
    resources: null
    type: kibana

```

...

- ❶ ❷ 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。インフラストラクチャーノードにテイントを追加した場合は、一致する容認も追加します。

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は、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は、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	v1.23.0
ip-10-0-139-146.us-east-2.compute.internal	Ready	master	60m	v1.23.0
ip-10-0-139-192.us-east-2.compute.internal	Ready	worker	51m	v1.23.0
ip-10-0-139-241.us-east-2.compute.internal	Ready	worker	51m	v1.23.0
ip-10-0-147-79.us-east-2.compute.internal	Ready	worker	51m	v1.23.0
ip-10-0-152-241.us-east-2.compute.internal	Ready	master	60m	v1.23.0
ip-10-0-139-48.us-east-2.compute.internal	Ready	infra	51m	v1.23.0

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-nkx17	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

5.7. 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ワーカー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。

Cluster Autoscaler は、コントロールプレーンノードを管理しない場合でも、クラスター内のすべてのノードのメモリー、CPU、および GPU の合計を計算します。これらの値は、単一マシン指向ではありません。これらは、クラスター全体での全リソースの集約です。たとえば、最大メモリーリソースの制限を設定する場合、Cluster Autoscaler は現在のメモリー使用量を計算する際にクラスター内のすべてのノードを含めます。この計算は、Cluster Autoscaler にワーカーリソースを追加する容量があるかどうかを判別するために使用されます。



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ルリングする可能性のあるコンピューター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 10 秒ごとに、クラスターで不要なノードをチェックし、それらを削除します。Cluster Autoscaler は、以下の条件が適用される場合に、ノードを削除すべきと考えます。

- ノードの使用率はクラスターの **ノード使用率レベル** のしきい値よりも低い場合。ノード使用率レベルとは、要求されたリソースの合計をノードに割り当てられたリソースで除算したものです。**ClusterAutoscaler** カスタムリソースで値を指定しない場合、Cluster Autoscaler は 50% の使用率に対応するデフォルト値 **0.5** を使用します。
- Cluster Autoscaler がノードで実行されているすべての Pod を他のノードに移動できる。Kubernetes スケジューラーは、ノード上の Pod のスケジュールを担当します。
- Cluster Autoscaler で、スケールダウンが無効にされたアノテーションがない。

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、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。

たとえば、CPU の上限を 64 コアに設定し、それぞれ 8 コアを持つマシンのみを作成するように Cluster Autoscaler を設定したとします。クラスターが 30 コアで起動する場合、Cluster Autoscaler は最大で 4 つのノード (合計 32 コア) を追加できます。この場合、総計は 62 コアになります。

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ようにしてください。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 を実行しないようにしてください。

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、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 をスケジューリングできますが、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。

クラスターの自動スケーリングは、マシン API が利用可能なプラットフォームでサポートされています。

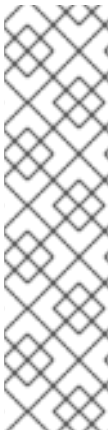
5.7.1. ClusterAutoscaler リソース定義

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 ①
  resourceLimits:
    maxNodesTotal: 24 ②
  cores:
    min: 8 ③
    max: 128 ④
  memory:
    min: 4 ⑤
    max: 256 ⑥
  gpus:
    - type: nvidia.com/gpu ⑦
      min: 0 ⑧
      max: 16 ⑨
    - type: amd.com/gpu
      min: 0
      max: 4
  scaleDown: ⑩
    enabled: true ⑪
    delayAfterAdd: 10m ⑫
    delayAfterDelete: 5m ⑬
    delayAfterFailure: 30s ⑭
    unneededTime: 5m ⑮
    utilizationThreshold: "0.4" ⑯
```

- ① 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 に割り当てられる **PriorityClass** の値と比較されます。
- ② 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。
- ③ クラスターにデプロイするコアの最小数を指定します。
- ④ クラスターにデプロイするコアの最大数を指定します。
- ⑤ クラスターのメモリーの最小量 (GiB 単位) を指定します。
- ⑥ クラスターのメモリーの最大量 (GiB 単位) を指定します。
- ⑦ オプション: 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。
- ⑧ クラスターにデプロイする GPU の最小数を指定します。

- 9 クラスターにデプロイする GPU の最大数を指定します。
- 10 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。
- 11 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。
- 12 オプション: 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。
- 13 オプション: ノードが最後に **削除** 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 **0s** が使用されます。
- 14 オプション: 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。
- 15 オプション: 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。
- 16 オプション: **ノード使用率レベル**を指定します。ノードの使用率がこの値を下回ると、不要なノードが削除の対象となります。ノード使用率は、要求されたリソースをそのノードに割り当てられたリソースで割ったもので、**"0"** より大きく **"1"** より小さい値でなければなりません。値を指定しない場合、Cluster Autoscaler は 50% の使用率に対応するデフォルト値 **"0.5"** を使用します。この値は文字列として表現する必要があります。



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。

Cluster Autoscaler がノードを管理しない場合でも、最小および最大の CPU、メモリー、および GPU の値は、クラスター内のすべてのノードのこれらのリソースを計算することによって決定されます。たとえば、Cluster Autoscaler がコントロールプレーンノードを管理しない場合でも、コントロールプレーンノードはクラスターのメモリーの合計に考慮されます。

5.7.2. 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。

5.8. 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ルリングできます。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

マシンをスケールリングするには、Cluster Autoscaler の Machine Autoscaler をデプロイする必要があります。Cluster Autoscaler は、スケール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ルリングできません。

5.8.1. 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 がスケール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、スケールリングするマシンセットの名前を指定するか、これを組み込みます。マシンセットの名前は、**<clusterid>-<machineset>-<region>** の形式を使用します。
- ❷ Cluster Autoscaler がクラスターのスケール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、Azure、RHOSP または vSphere 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。

特殊なワークロードに使用されるコストがかかり、用途が限られたハードウェアを稼働する場合などのユースケースにはこの値を **0** に設定するか、若干大きいマシンを使用してマシンセットをスケールリングすることで、コストを節約できます。Cluster Autoscaler は、マシンが使用されていない場合にマシンセットをゼロにスケールダウンします。



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 OpenShift Container Platform インストールプロセス時に作成される 3 つのコンピューターマシンセットについては、**spec.minReplicas** の値を **0** に設定しないでください。

- 3 Cluster Autoscaler がクラスタースケージングの開始後に指定された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。
- 4 このセクションでは、スケージ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。
- 5 **kind** パラメーターの値は常に **MachineSet** です。
- 6 **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。

5.8.2. 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。

5.9. 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。

5.9.1. 機能ゲートについて

FeatureGate カスタムリソース (CR) を使用して、クラスタ内の特定の機能セットを有効にすることができます。機能セットは、デフォルトで有効にされない OpenShift Container Platform 機能のコレクションです。

FeatureGate CR を使用して、以下の機能セットをアクティブにすることができます。

- **TechPreviewNoUpgrade.** この機能セットは、現在のテクノロジープレビュー機能のサブセットです。この機能セットにより、実稼働クラスタではこれらのテクノロジープレビュー機能が無効にし、テストクラスタで機能を有効にして十分にテストを行うことができます。この機能セットを有効にすると元に戻すことができなくなり、マイナーバージョン更新ができなくなります。この機能セットは、実稼働クラスタでは推奨されません。



警告

クラスターで **TechPreviewNoUpgrade** 機能セットを有効にすると、元に戻すことができず、マイナーバージョンの更新が妨げられます。本番クラスターでは、この機能セットを有効にしないでください。

この機能セットにより、以下のテクノロジープレビュー機能が有効になります。

- Microsoft Azure File CSI Driver Operator: Microsoft Azure File Storage に Container Storage Interface (CSI) ドライバーを使用して、永続ボリューム (PV) のプロビジョニングを有効にします。
- CSI の自動移行: サポートされているインツリーのボリュームプラグインを等価な Container Storage Interface (CSI) ドライバーに自動的に移行できます。サポート対象:
  - Amazon Web Services (AWS) Elastic Block Storage (EBS)
  - OpenStack Cinder
  - Azure Disk
  - Azure File
  - Google Cloud Platform Persistent Disk (CSI)
  - VMware vSphere
- Cluster Cloud Controller Manager Operator: インツリーのクラウドコントローラーではなく、Cluster Cloud Controller Manager Operator を有効にします。テクノロジープレビューとして利用可能な対象は以下のとおりです。
  - Alibaba Cloud
  - Amazon Web Services (AWS)
  - Google Cloud Platform (GCP)
  - IBM Cloud
  - Microsoft Azure
  - Red Hat OpenStack Platform (RHOSP)
  - VMware vSphere
- 共有リソース CSI ドライバー:
- OpenShift Container Platform ビルドシステムに対する CSI ボリュームのサポート
- ノード上のスワップメモリー

5.9.2. Web コンソールで機能セットの有効化

FeatureGate カスタムリソース (CR) を編集して、OpenShift Container Platform Web コンソールを使用してクラスター内のすべてのノードの機能セットを有効にすることができます。

手順

機能セットを有効にするには、以下を実行します。

1. OpenShift Container Platform Web コンソールで、**Administration → Custom Resource Definitions** ページに切り替えます。
2. **Custom Resource Definitions** ページで、**FeatureGate** をクリックします。
3. **Custom Resource Definition Details** ページで、**Instances** タブをクリックします。
4. **cluster** 機能ゲートをクリックしてから、**YAML** タブをクリックします。
5. **cluster** インスタンスを編集して特定の機能セットを追加します。



警告

クラスターで **TechPreviewNoUpgrade** 機能セットを有効にすると、元に戻すことができず、マイナーバージョンの更新が妨げられます。本番クラスターでは、この機能セットを有効にしないでください。

機能ゲートカスタムリソースのサンプル

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster 1
# ...
spec:
  featureSet: TechPreviewNoUpgrade 2
```

1 **FeatureGate** CR の名前は **cluster** である必要があります。

2 有効にする機能セットを追加します。

- **TechPreviewNoUpgrade** は、特定のテクノロジープレビュー機能を有効にします。

変更を保存すると、新規マシン設定が作成され、マシン設定プールが更新され、変更が適用されている間に各ノードのスケジューリングが無効になります。

検証

ノードが Ready 状態に戻ると、ノードの **kubelet.conf** ファイルを確認して機能ゲートが有効になっていることを確認できます。

1. Web コンソールの **Administrator** パースペクティブで、**Compute → Nodes** に移動します。
2. ノードを選択します。

3. **Node details** ページで **Terminal** をクリックします。
4. ターミナルウィンドウで、root ディレクトリーを **/host** に切り替えます。

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

5. **kubelet.conf** ファイルを表示します。

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

出力例

```
# ...  
featureGates:  
  InsightsOperatorPullingSCA: true,  
  LegacyNodeRoleBehavior: false  
# ...
```

true として一覧表示されている機能は、クラスターで有効になっています。



注記

一覧表示される機能は、OpenShift Container Platform のバージョンによって異なります。

5.9.3. CLI を使用した機能セットの有効化

FeatureGate カスタムリソース (CR) を編集し、OpenShift CLI (**oc**) を使用してクラスター内のすべてのノードの機能セットを有効にすることができます。

前提条件

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

手順

機能セットを有効にするには、以下を実行します。

1. **cluster** という名前の **FeatureGate** CR を編集します。

```
$ oc edit featuregate cluster
```



警告

クラスターで **TechPreviewNoUpgrade** 機能セットを有効にすると、元に戻すことができず、マイナーバージョンの更新が妨げられます。本番クラスターでは、この機能セットを有効にしないでください。

FeatureGate カスタムリソースのサンプル

```

apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster ❶
# ...
spec:
  featureSet: TechPreviewNoUpgrade ❷

```

❶ **FeatureGate** CR の名前は **cluster** である必要があります。

❷ 有効にする機能セットを追加します。

- **TechPreviewNoUpgrade** は、特定のテクノロジープレビュー機能を有効にします。

変更を保存すると、新規マシン設定が作成され、マシン設定プールが更新され、変更が適用されている間に各ノードのスケジューリングが無効になります。

検証

ノードが Ready 状態に戻ると、ノードの **kubelet.conf** ファイルを確認して機能ゲートが有効になっていることを確認できます。

1. Web コンソールの **Administrator** パースペクティブで、**Compute → Nodes** に移動します。
2. ノードを選択します。
3. **Node details** ページで **Terminal** をクリックします。
4. ターミナルウィンドウで、root ディレクトリーを **/host** に切り替えます。

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

5. **kubelet.conf** ファイルを表示します。

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

出力例

```

# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...

```

true として一覧表示されている機能は、クラスターで有効になっています。



注記

一覧表示される機能は、OpenShift Container Platform のバージョンによって異なります。

5.10. 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。

5.10.1. 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レイヤー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ない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。



注記

etcd 暗号化は、キーではなく、値のみを暗号化します。リソースの種類、namespace、およびオブジェクト名は暗号化されません。

バックアップ中に etcd 暗号化が有効になっている場合は、**static_kuberesources_<timestamp>.tar.gz** ファイルに etcd スナップショットの暗号化キーが含まれています。セキュリティ上の理由から、このファイルは etcd スナップショットとは別に保存してください。ただし、このファイルは、それぞれの etcd スナップショットから etcd の以前の状態を復元するために必要です。

5.10.2. 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

初期暗号化プロセスが完了するまで、etcd リソースをバックアップしないでください。暗号化プロセスが完了しない場合、バックアップは一部のみ暗号化される可能性があります。

etcd 暗号化を有効にすると、いくつかの変更が発生する可能性があります。

- etcd 暗号化は、いくつかのリソースのメモリー消費に影響を与える可能性があります。
- リーダーがバックアップを提供する必要があるため、バックアップのパフォーマンスに一時的な影響が生じる場合があります。
- ディスク I/O は、バックアップ状態を受け取るノードに影響を与え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 1
```

- 1 **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
```

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。

- c. OpenShift OAuth 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 authentication.operator.openshift.io -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: 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。

5.10.3. 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 ❶
```

❶ **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。

- c. 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復号化されたことを確認します。

```
$ oc get authentication.operator.openshift.io -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。

5.10.4. 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

単一のコントロールプレーンホストからのバックアップのみを保存します。クラスター内の各コントロールプレーンホストからのバックアップは取得しないでください。

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. コントロールプレーン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 **/host** に変更します。

```
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

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
```

```
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created
temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-
25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream;
downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching
snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-
25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read;
closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched
snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved",
"path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

この例では、コントロールプレーンホストの **/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。 **cluster-backup.sh** スクリプトで、その有効性を確認します。
- **static_kubernetes_<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。

5.10.5. etcd データのデフラグ

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

これらの主要な指標をモニターします。

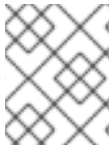
- **etcd_server_quota_backend_bytes**、これは現在のクォータ制限です
- **etcd_mvcc_db_total_size_in_use_in_bytes**、これはヒストリーコンパクション後の実際のデータベース使用状況を示します。

- **etcd_mvcc_db_total_size_in_bytes** はデフラグ待ちの空き領域を含むデータベースサイズを表します。

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

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

デフラグは自動的に行われますが、手動でトリガーすることもできます。



注記

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

5.10.5.1. 自動デフラグ

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

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

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



警告

自動デフラグにより、Kubernetes コントローラーマネージャーなどのさまざまな OpenShift コアコンポーネントでリーダー選出の失敗が発生し、失敗したコンポーネントの再起動がトリガーされる可能性があります。再起動は無害であり、次に実行中のインスタンスへのフェイルオーバーをトリガーするか、再起動後にコンポーネントが再び作業を再開します。

最適化が成功した場合のログ出力の例

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

最適化に失敗した場合のログ出力の例

```
failed defrag on member: <member_name>, memberID: <member_id>: <error_message>
```

5.10.5.2. 手動デフラグ

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

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

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



警告

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

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

前提条件

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

手順

1. リーダーを最後にデフラグする必要があるため、どの etcd メンバーがリーダーであるかを判別します。
  - a. etcd Pod の一覧を取得します。

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

出力例

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

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

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

出力例

Defaulting container name to etcdctl.

Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see all of the containers in this pod.

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

```

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

- e. これらの手順を繰り返して他の etcd メンバーのそれぞれに接続し、デフラグします。常に最後にリーダーをデフラグします。
etcd Pod が回復するように、デフラグアクションごとに 1 分以上待機します。etcd Pod が回復するまで、etcd メンバーは応答しません。
3. 領域のクォータの超過により **NOSPACE** アラームがトリガーされる場合、それらをクリアします。
  - a. **NOSPACE** アラームがあるかどうかを確認します。

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

出力例

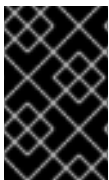
```
memberID:12345678912345678912 alarm:NOSPACE
```

- b. アラームをクリアします。

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

5.10.6. クラスターの直前の状態への復元

保存された etcd のバックアップを使用して、クラスターの以前の状態を復元したり、大多数のコントロールプレーンホストが失われたクラスターを復元したりできます。



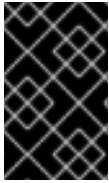
重要

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 4.7.2 クラスターは、4.7.2 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

- インストール時に使用したものと同様、証明書ベースの **kubeconfig** ファイルを介して、**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します。
- リカバリーホストとして使用する正常なコントロールプレーンホストがあること。
- コントロールプレーンホストへの 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_<datetimestamp>.db** および **static_kuberesources_<datetimestamp>.tar.gz** の形式にする必要があります。

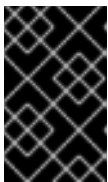


重要

非復元コントロールプレーンノードの場合は、SSH 接続を確立したり、静的 Pod を停止したりする必要はありません。他のリカバリー以外のコントロールプレーンマシンを 1 つずつ削除し、再作成します。

手順

- 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。
- 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コントロールプレーン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。

- 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。
- 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静的 Pod を停止する必要はありません。

- 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。
- 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。

```
$ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

- etcd Pod が停止していることを確認します。

```
$ 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。

- 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。

-

```
$ 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。

```
$ 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。

```
$ sudo mv /var/lib/etcd/ /tmp
```

- g. リカバリーホストではない他のコントロールプレーン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。

```
$ 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
```



注記

最後の etcd バックアップの後にノード証明書が更新された場合、復元プロセスによってノードが **NotReady** 状態になる可能性があります。

8. ノードをチェックして、**Ready** 状態であることを確認します。

- a. 以下のコマンドを実行します。

```
$ oc get nodes -w
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
host-172-25-75-28	Ready	master	3d20h	v1.23.3+e419edf
host-172-25-75-38	Ready	infra,worker	3d20h	v1.23.3+e419edf
host-172-25-75-40	Ready	master	3d20h	v1.23.3+e419edf
host-172-25-75-65	Ready	master	3d20h	v1.23.3+e419edf
host-172-25-75-74	Ready	infra,worker	3d20h	v1.23.3+e419edf
host-172-25-75-79	Ready	worker	3d20h	v1.23.3+e419edf
host-172-25-75-86	Ready	worker	3d20h	v1.23.3+e419edf
host-172-25-75-98	Ready	infra,worker	3d20h	v1.23.3+e419edf

すべてのノードが状態を報告するのに数分かかる場合があります。

- b. **NotReady** 状態のノードがある場合は、ノードにログインし、各ノードの `/var/lib/kubelet/pki` ディレクトリーからすべての PEM ファイルを削除します。ノードに SSH 接続するか、Web コンソールのターミナルウィンドウを使用できます。

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

サンプル pki ディレクトリー

```
sh-4.4# pwd
/var/lib/kubelet/pki
sh-4.4# ls
kubelet-client-2022-04-28-11-24-09.pem kubelet-server-2022-04-28-11-24-15.pem
kubelet-client-current.pem           kubelet-server-current.pem
```

9. すべてのコントロールプレーンホストで 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。

```
$ sudo systemctl restart kubelet.service
```

- b. 他のすべてのコントロールプレーンホストでこの手順を繰り返します。

10. 保留中の CSR を承認します。

- a. 現在の CSR の一覧を取得します。

```
$ oc get csr
```

出力例

```
NAME      AGE  SIGNERNAME                                REQUESTOR
CONDITION
csr-2s94x  8m3s  kubernetes.io/kubelet-serving             system:node:<node_name>
Pending  ❶
csr-4bd6t  8m3s  kubernetes.io/kubelet-serving             system:node:<node_name>
Pending  ❷
csr-4hl85  13m   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper Pending
❸
csr-zh8hp  3m8s  kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper Pending
❹
...
```

❶ ❶ ❷ 保留中の kubelet サービス CSR (ユーザーがプロビジョニングしたインストール用)。

❸ ❹ 保留中の **node-bootstrapper** CSR。

b. CSR の詳細をレビューし、これが有効であることを確認します。

```
$ oc describe csr <csr_name> ❶
```

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

c. それぞれの有効な **node-bootstrapper** CSR を承認します。

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

d. ユーザーによってプロビジョニングされるインストールの場合は、それぞれの有効な kubelet 提供の CSR を承認します。

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

11. 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。

```
$ sudo crictl ps | grep etcd | egrep -v "operator|etcd-guard"
```

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。

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

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。



注記

次の手順は、**OVN**Kubernetes Container Network Interface (CNI) プラグインを使用している場合にのみ実行してください。

12. すべてのホストで Open Virtual Network (OVN) Kubernetes Pod を再起動します。
- a. ノースバウンドデータベース (nbdb) とサウスバウンドデータベース (sbdb) を削除します。Secure Shell (SSH) を使用してリカバリーホストと残りのコントロールプレーンノードにアクセスし、次のコマンドを実行します。

```
$ sudo rm -f /var/lib/ovn/etc/*.db
```

- b. 次のコマンドを実行して、すべての OVN-Kubernetes コントロールプレーン Pod を削除します。

```
$ oc delete pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

- c. 次のコマンドを実行して、OVN-Kubernetes コントロールプレーン Pod が再度デプロイされ、**Running** 状態になっていることを確認します。

```
$ oc get pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
ovnkube-master-nb24h	4/4	Running	0	48s

- d. 次のコマンドを実行して、すべての **ovnkube-node** Pod を削除します。

```
$ oc get pods -n openshift-ovn-kubernetes -o name | grep ovnkube-node | while read p ;  
do oc delete $p -n openshift-ovn-kubernetes ; done
```

- e. 次のコマンドを実行して、すべての **ovnkube-node** Pod が再度デプロイされ、**Running** 状態になっていることを確認します。

```
$ oc get pods -n openshift-ovn-kubernetes | grep ovnkube-node
```


13. 他の非復旧のコントロールプレーンマシンを1つずつ削除して再作成します。マシンが再作成された後、新しいリビジョンが強制され、etcd が自動的にスケールアップします。
- ユーザーがプロビジョニングしたベアメタルインストールを使用する場合は、最初に作成したときと同じ方法を使用して、コントロールプレーンマシンを再作成できます。詳細については、ユーザーがプロビジョニングしたクラスターをベアメタルにインストールするを参照してください。

**警告**

リカバリーホストのマシンを削除し、再作成しないでください。

- インストーラーでプロビジョニングされるインフラストラクチャーを実行している場合、またはマシン API を使用してマシンを作成している場合は、以下の手順を実行します。

**警告**

リカバリーホストのマシンを削除し、再作成しないでください。

インストーラーによってプロビジョニングされたインフラストラクチャーでのベアメタルインストールの場合、コントロールプレーンマシンは再作成されません。詳細については、ベアメタルコントロールプレーンノードの交換を参照してください。

- a. 失われたコントロールプレーンホストのいずれかのマシンを取得します。
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 machines -n openshift-machine-api -o wide
```

出力例:

NAME NODE	PHASE PROVIDERID	TYPE	REGION STATE	ZONE	AGE
clustername-8qw5l-master-0		Running	m4.xlarge	us-east-1	us-east-1a
3h37m ip-10-0-131-183.ec2.internal	aws:///us-east-1a/i-0ec2782f8287dfb7e				
stopped 1					
clustername-8qw5l-master-1		Running	m4.xlarge	us-east-1	us-east-1b
3h37m ip-10-0-143-125.ec2.internal	aws:///us-east-1b/i-096c349b700a19631				
running					
clustername-8qw5l-master-2		Running	m4.xlarge	us-east-1	us-east-1c
3h37m ip-10-0-154-194.ec2.internal	aws:///us-east-1c/i-02626f1dba9ed5bba				
running					
clustername-8qw5l-worker-us-east-1a-wbtgd		Running	m4.large	us-east-1	us-east-1a
3h28m ip-10-0-129-226.ec2.internal	aws:///us-east-1a/i-				

```
010ef6279b4662ced running
clustername-8qw5l-worker-us-east-1b-lrdxb Running m4.large us-east-1 us-
east-1b 3h28m ip-10-0-144-248.ec2.internal aws:///us-east-1b/i-
0cb45ac45a166173b running
clustername-8qw5l-worker-us-east-1c-pkg26 Running m4.large us-east-1 us-
east-1c 3h28m ip-10-0-170-181.ec2.internal aws:///us-east-1c/i-
06861c00007751b0a running
```

- 1 これは、失われたコントロールプレーンホストのコントロールプレーンマシンです (**ip-10-0-131-183.ec2.internal**)。

b. マシン設定をファイルシステムのファイルに保存します。

```
$ oc get machine clustername-8qw5l-master-0 \ 1
-n openshift-machine-api \
-o yaml \
> new-master-machine.yaml
```

- 1 失われたコントロールプレーンホストのコントロールプレーンマシンの名前を指定します。

c. 直前の手順で作成された **new-master-machine.yaml** ファイルを編集し、新しい名前を割り当て、不要なフィールドを削除します。

i. **status** セクション全体を削除します。

```
status:
  addresses:
    - address: 10.0.131.183
      type: InternalIP
    - address: ip-10-0-131-183.ec2.internal
      type: InternalDNS
    - address: ip-10-0-131-183.ec2.internal
      type: Hostname
  lastUpdated: "2020-04-20T17:44:29Z"
  nodeRef:
    kind: Node
    name: ip-10-0-131-183.ec2.internal
    uid: acca4411-af0d-4387-b73e-52b2484295ad
  phase: Running
  providerStatus:
    apiVersion: awsproviderconfig.openshift.io/v1beta1
    conditions:
      - lastProbeTime: "2020-04-20T16:53:50Z"
        lastTransitionTime: "2020-04-20T16:53:50Z"
        message: machine successfully created
        reason: MachineCreationSucceeded
        status: "True"
        type: MachineCreation
    instanceId: i-0fdb85790d76d0c3f
    instanceState: stopped
    kind: AWSMachineProviderStatus
```

ii. **metadata.name** フィールドを新規の名前に変更します。

古いマシンと同じベース名を維持し、最後の番号を次に利用可能な番号に変更することが推奨されます。この例では、**clustername-8qw5l-master-0** は **clustername-8qw5l-master-3** に変更されています。

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  name: clustername-8qw5l-master-3
  ...
```

- iii. **spec.providerID** フィールドを削除します。

```
providerID: aws:///us-east-1a/i-0fdb85790d76d0c3f
```

- iv. **metadata.annotations** および **metadata.generation** フィールドを削除します。

```
annotations:
  machine.openshift.io/instance-state: running
  ...
generation: 2
```

- v. **metadata.resourceVersion** および **metadata.uid** フィールドを削除します。

```
resourceVersion: "13291"
uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

- d. 失われたコントロールプレーンホストのマシンを削除します。

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 ❶
```

- ❶ 失われたコントロールプレーンホストのコントロールプレーンマシンの名前を指定します。

- e. マシンが削除されたことを確認します。

```
$ oc get machines -n openshift-machine-api -o wide
```

出力例:

NAME	PHASE	TYPE	REGION	ZONE	AGE
NODE	PROVIDERID	STATE			
clustername-8qw5l-master-1	Running	m4.xlarge	us-east-1	us-east-1b	3h37m
ip-10-0-143-125.ec2.internal	aws:///us-east-1b/i-096c349b700a19631	running			
clustername-8qw5l-master-2	Running	m4.xlarge	us-east-1	us-east-1c	3h37m
ip-10-0-154-194.ec2.internal	aws:///us-east-1c/i-02626f1dba9ed5bba	running			
clustername-8qw5l-worker-us-east-1a-wbtgd	Running	m4.large	us-east-1	us-east-1a	3h28m
ip-10-0-129-226.ec2.internal	aws:///us-east-1a/i-010ef6279b4662ced	running			
clustername-8qw5l-worker-us-east-1b-lrdxb	Running	m4.large	us-east-1	us-east-1b	3h28m
ip-10-0-144-248.ec2.internal	aws:///us-east-1b/i-				

```
0cb45ac45a166173b running
clustername-8qw5l-worker-us-east-1c-pkg26 Running m4.large us-east-1 us-
east-1c 3h28m ip-10-0-170-181.ec2.internal aws:///us-east-1c/i-
06861c00007751b0a running
```

- f. **new-master-machine.yaml** ファイルを使用してマシンを作成します。

```
$ oc apply -f new-master-machine.yaml
```

- g. 新規マシンが作成されたことを確認します。

```
$ oc get machines -n openshift-machine-api -o wide
```

出力例:

NAME	PHASE	TYPE	REGION	ZONE
clustername-8qw5l-master-1	Running	m4.xlarge	us-east-1	us-east-1b
clustername-8qw5l-master-2	Running	m4.xlarge	us-east-1	us-east-1c
clustername-8qw5l-master-3	Provisioning	m4.xlarge	us-east-1	us-east-1a
clustername-8qw5l-worker-us-east-1a-wbtgd	Running	m4.large	us-east-1	us-east-1a
clustername-8qw5l-worker-us-east-1b-lrdxb	Running	m4.large	us-east-1	us-east-1b
clustername-8qw5l-worker-us-east-1c-pkg26	Running	m4.large	us-east-1	us-east-1c

- 1 新規マシン **clustername-8qw5l-master-3** が作成され、**Provisioning** から **Running** にフェーズが変更されると準備状態になります。

新規マシンが作成されるまでに数分の時間がかかる場合があります。etcd クラスター Operator はマシンまたはノードが正常な状態に戻ると自動的に同期します。

- h. リカバリーホストではない喪失したコントロールプレーンホストで、これらのステップを繰り返します。

14. 次のコマンドを入力して、クォーラムガードをオフにします。

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": {"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

このコマンドにより、シークレットを正常に再作成し、静的 Pod をロールアウトできるようになります。

15. リカバリーホスト内の別のターミナルウィンドウで、次のコマンドを実行してリカバリー **kubeconfig** ファイルをエクスポートします。

```
$ export KUBECONFIG=/etc/kubernetes/static-pod-resources/kube-apiserver-
certs/secrets/node-kubeconfigs/localhost-recovery.kubeconfig
```

16. etcd の再デプロイメントを強制的に実行します。
リカバリー **kubeconfig** ファイルをエクスポートしたのと同じターミナルウィンドウで、次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。

17. 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。

18. 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. Kubernetes API サーバーの新規ロールアウトを強制的に実行します。

```
$ 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リース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リース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 ❶
```

❶ 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、再試行します。

- b. Kubernetes コントローラマネージャーの新規ロールアウトを強制的に実行します。

```
$ 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リース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リース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 ❶
```

❶ 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、再試行します。

- c. Kubernetes スケジューラーの新規ロールアウトを強制的に実行します。

```
$ 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リース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リース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 ❶
```

❶ 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。

19. すべてのコントロールプレーンホストが起動しており、クラスタに参加していることを確認します。
クラスタ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 -n openshift-etcd get pods -l k8s-app=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
```

復元手順の後にすべてのワークロードが通常の動作に戻るようには、Kubernetes API 情報を保存している各 Pod を再起動します。これには、ルーター、Operator、サードパーティーコンポーネントなどの OpenShift Container Platform コンポーネントが含まれます。

注記

前の手順が完了したら、すべてのサービスが復元された状態に戻るまで数分間待つ必要がある場合があります。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。

即時認証に **system:admin kubeconfig** ファイルを使用することを検討してください。この方法は、OAuth トークンではなく SSL/TLS クライアント証明書に基づいて認証を行います。以下のコマンドを実行し、このファイルを使用して認証できます。

```
$ export KUBECONFIG=<installation_directory>/auth/kubeconfig
```

以下のコマンドを実行して、認証済みユーザー名を表示します。

```
$ oc whoami
```

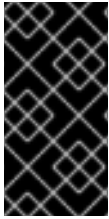
関連情報

- [ユーザーによってプロビジョニングされるクラスタのベアメタルへのインストール](#)
- [ベアメタルコントロールプレーンノードの交換](#)

5.10.7. 永続ストレージの状態復元に関する問題および回避策

OpenShift Container Platform クラスタがいずれかの形式の永続ストレージを使用する場合に、クラスタの状態は通常 etcd 外に保存されます。たとえば、Pod で実行されている Elasticsearch クラス

ター、または **StatefulSet** オブジェクトで実行されているデータベースなどである可能性があります。etcd バックアップから復元する場合には、OpenShift Container Platform のワークロードのステータスも復元されます。ただし、etcd スナップショットが古い場合には、ステータスは無効または期限切れの可能性もあります。



重要

永続ボリューム (PV) の内容は etcd スナップショットには含まれません。etcd スナップショットから OpenShift Container Platform クラスタを復元する時に、重要ではないワークロードから重要なデータにアクセスしたり、その逆ができたりする場合があります。

以下は、古いステータスを生成するシナリオ例です。

- MySQL データベースが PV オブジェクトでバックアップされる Pod で実行されている。etcd スナップショットから OpenShift Container Platform を復元すると、Pod の起動を繰り返し試行しても、ボリュームをストレージプロバイダーに戻したり、実行中の MySQL Pod が生成したりされるわけではありません。この Pod は、ストレージプロバイダーでボリュームを復元し、次に PV を編集して新規ボリュームを参照するように手動で復元する必要があります。
- Pod P1 は、ノード X に割り当てられているボリューム A を使用している。別の Pod がノード Y にある同じボリュームを使用している場合に etcd スナップショットが作成された場合に、etcd の復元が実行されると、ボリュームがノード Y に割り当てられていることが原因で Pod P1 が正常に起動できなくなる可能性があります。OpenShift Container Platform はこの割り当てを認識せず、ボリュームが自動的に切り離されるわけではありません。これが生じる場合には、ボリュームをノード Y から手動で切り離し、ノード X に割り当ててすることで Pod P1 を起動できるようにします。
- クラウドプロバイダーまたはストレージプロバイダーの認証情報が etcd スナップショットの作成後に更新された。これが原因で、プロバイダーの認証情報に依存する CSI ドライバーまたは Operator が機能しなくなります。これらのドライバーまたは Operator で必要な認証情報を手動で更新する必要がある場合があります。
- デバイスが etcd スナップショットの作成後に OpenShift Container Platform ノードから削除されたか、名前が変更された。ローカルストレージ Operator で、**/dev/disk/by-id** または **/dev** ディレクトリーから管理する各 PV のシンボリックリンクが作成されます。この状況では、ローカル PV が存在しないデバイスを参照してしまう可能性があります。この問題を修正するには、管理者は以下を行う必要があります。
  1. デバイスが無効な PV を手動で削除します。
  2. 各ノードからシンボリックリンクを削除します。
  3. **LocalVolume** または **LocalVolumeSet** オブジェクトを削除します (ストレージ → 永続ストレージの設定 → ローカルボリュームを使用した永続ストレージ → ローカルストレージ Operator のリソースの削除 を参照)。

5.11. 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。

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

Pod 中断バジェット では、メンテナンスのためのノードのドレインなど、運用中の Pod に対する安全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

Available は、**Ready=True** の状態にある Pod 数を指します。**ready=True** は、要求に対応でき、一致するすべてのサービスの負荷分散プールに追加する必要がある Pod を指します。

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	MAX UNAVAILABLE
ALLOWED DISRUPTIONS	AGE		
openshift-apiserver	openshift-apiserver-pdb	N/A	1
121m			1
openshift-cloud-controller-manager	aws-cloud-controller-manager	1	N/A
125m			1
openshift-cloud-credential-operator	pod-identity-webhook	1	N/A
117m			1
openshift-cluster-csi-drivers	aws-ebs-csi-driver-controller-pdb	N/A	1
121m			1
openshift-cluster-storage-operator	csi-snapshot-controller-pdb	N/A	1
122m			1
openshift-cluster-storage-operator	csi-snapshot-webhook-pdb	N/A	1
122m			1
openshift-console	console	N/A	1
116m			1
#...			

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レショ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。

5.11.2. Pod の Disruption Budget を使用して起動している 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** オブジェクトを使用して指定します。

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/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 2
  selector: 3
    matchLabels:
      name: my-pod
```

- 1** **PodDisruptionBudget** は **policy/v1** 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。プロジェクト内のすべての Pod を選択するには、このパラメーターを空白のままにします (例: **selector {}**)。

または、以下を実行します。

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      name: my-pod
```

- 1** **PodDisruptionBudget** は **policy/v1** 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。プロジェクト内のすべての Pod を選択するには、このパラメーターを空白のままにします (例: **selector {}**)。

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>
```

5.12. 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。

5.12.1. Cloud Credential Operator ユーティリティーを使用したクラウドプロバイダー資格情報のローテーション

Cloud Credential Operator (CCO) ユーティリティー **ccctl** は、IBM Cloud にインストールされたクラスターのシークレットの更新をサポートしています。

5.12.1.1. IBM Cloud の API キーのローテーション

既存のサービス ID の API キーをローテーションし、対応するシークレットを更新できます。

前提条件

- **ccctl** バイナリーを設定している。
- IBM Cloud にインストールされたライブ OpenShift Container Platform クラスターに既存のサービス ID がある。

手順

- **ccctl** ユーティリティーを使用して、サービス ID の API キーをローテーションし、シークレットを更新します。

```
$ ccctl ibmcloud refresh-keys \
  --kubeconfig <openshift_kubeconfig_file> \ ❶
  --credentials-requests-dir <path_to_credential_requests_directory> \ ❷
  --name <name> ❸
```

- ❶ クラスターに関連付けられている **kubeconfig** ファイル。たとえば、**<installation_directory>/auth/kubeconfig** です。
- ❷ 認証情報の要求が保存されるディレクトリー。
- ❸ OpenShift Container Platform クラスターの名前。



注記

クラスターで **TechPreviewNoUpgrade** 機能セットによって有効化されたテクノロジープレビュー機能を使用している場合は、**--enable-tech-preview** パラメーターを含める必要があります。

5.12.2. クラウドプロバイダーの認証情報の手動によるローテーション

クラウドプロバイダーの認証情報が何らかの理由で変更される場合、クラウドプロバイダーの認証情報の管理に Cloud Credential Operator (CCO) が使用するシークレットを手動で更新する必要があります。

クラウド認証情報をローテーションするプロセスは、CCO を使用するよう設定されているモードによって変わります。mint モードを使用しているクラスターの認証情報をローテーションした後に、削除された認証情報で作成されたコンポーネントの認証情報は手動で削除する必要があります。

前提条件

- クラスターは、使用している CCO モードでのクラウド認証情報の手動ローテーション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。
  - mint モードについては、Amazon Web Services (AWS) および Google Cloud Platform (GCP) がサポートされます。
  - passthrough モードは、Amazon Web Services (AWS)、Microsoft Azure、Google Cloud Platform (GCP)、Red Hat OpenStack Platform (RHOSP)、Red Hat Virtualization (RHV)、および VMware vSphere でサポートされます。
- クラウドプロバイダーとのインターフェイスに使用される認証情報を変更している。
- 新規認証情報には、モードの CCO がクラスターで使用されるように設定するのに十分なパーミッションがある。

手順

- 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。
- Secrets** ページの表で、クラウドプロバイダーの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	aws-creds
Azure	azure-credentials
GCP	gcp-credentials
RHOSP	openstack-credentials
RHV	ovirt-credentials
VMware vSphere	vsphere-creds

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、**Edit Secret** を選択します。
4. **Value** フィールドの内容を記録します。この情報を使用して、認証情報の更新後に値が異なることを確認できます。
5. **Value** フィールドのテキストをクラウドプロバイダーの新規の認証情報で更新し、**Save** をクリックします。
6. vSphere CSI Driver Operator が有効になっていない vSphere クラスターの認証情報を更新する場合は、Kubernetes コントローラマネージャーを強制的にロールアウトして更新された認証情報を適用する必要があります。



注記

vSphere CSI Driver Operator が有効になっている場合、この手順は不要です。

更新された vSphere 認証情報を適用するには、**cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインし、以下のコマンドを実行します。

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

認証情報がロールアウトされている間、Kubernetes Controller Manager Operator のステータスは **Progressing=true** を報告します。ステータスを表示するには、次のコマンドを実行します。

```
$ oc get co kube-controller-manager
```

7. クラスターの CCO が mint モードを使用するように設定されている場合、個別の **CredentialsRequest** オブジェクトによって参照される各コンポーネントシークレットを削除します。
  - a. **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform CLI にログインします。
  - b. 参照されたすべてのコンポーネントシークレットの名前および namespace を取得します。

```
$ oc -n openshift-cloud-credential-operator get CredentialsRequest \
  -o json | jq -r '.items[] | select (.spec.providerSpec.kind==\"<provider_spec>\") | \
  .spec.secretRef'
```

ここで、**<provider_spec>** はクラウドプロバイダーの対応する値になります。

- AWS: **AWSProviderSpec**
- GCP: **GCPProviderSpec**

AWS の部分的なサンプル出力

```
{
  \"name\": \"ebs-cloud-credentials\",
```

```
"namespace": "openshift-cluster-csi-drivers"
}
{
  "name": "cloud-credential-operator-iam-ro-creds",
  "namespace": "openshift-cloud-credential-operator"
}
```

- c. 参照されるコンポーネントの各シークレットを削除します。

```
$ oc delete secret <secret_name> \ ❶
-n <secret_namespace> ❷
```

❶ シークレットの名前を指定します。

❷ シークレットを含む namespace を指定します。

AWS シークレットの削除例

```
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

プロバイダーコンソールから認証情報を手動で削除する必要はありません。参照されるコンポーネントのシークレットを削除すると、CCO はプラットフォームから既存の認証情報を削除し、新規の認証情報を作成します。

検証

認証情報が変更されたことを確認するには、以下を実行します。

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. **Value** フィールドの内容が変更されていることを確認します。

関連情報

- [vSphere CSI Driver Operator](#)

5.12.3. クラウドプロバイダーの認証情報の削除

Cloud Credential Operator (CCO) を mint モードで使用して OpenShift Container Platform クラスターをインストールした後に、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。サポート対象プラットフォームは AWS および GCP。

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、クラウドプロバイダーの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	aws-creds
GCP	gcp-credentials

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。

関連情報

- [Cloud Credential Operator について](#)
- [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](#)

5.13. 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。

5.13.1. ミラーリングの Cluster Samples Operator のサポート

インストール時に、OpenShift Container Platform は **imagestreamtag-to-image** という名前の設定マップを **openshift-cluster-samples-operator** namespace に作成します。**imagestreamtag-to-image** 設定マップには、各イメージストリームタグのエントリー (設定されるイメージ) が含まれます。

設定マップの data フィールドの各エントリーのキーの形式は、**<image_stream_name>_<image_stream_tag_name>** です。

OpenShift Container Platform の非接続インストール時に、Cluster Samples Operator のステータスは **Removed** に設定されます。これを **Managed** に変更することを選択する場合、サンプルがインストールされます。



注記

ネットワークが制限されている環境または切断されている環境でサンプルを使用するには、ネットワークの外部のサービスにアクセスする必要がある場合があります。サービスの例には、Github、Maven Central、npm、RubyGems、PyPi などがあります。クラスターサンプルオペレーターのオブジェクトが必要なサービスに到達できるようにするために、追加の手順を実行する必要がある場合があります。

この設定マップを、イメージストリームがインポートできるようにミラーリングする必要のあるイメージについての参照情報として使用できます。

- Cluster Samples Operator が **Removed** に設定される場合、ミラーリングされたレジストリーを作成するか、使用する必要のある既存のミラーリングされたレジストリーを判別できます。
- 新しい設定マップをガイドとして使用し、ミラーリングされたレジストリーに必要なサンプルをミラーリングします。
- Cluster Samples Operator 設定オブジェクトの **skippedImagestreams** 一覧に、ミラーリングされていないイメージストリームを追加します。
- Cluster Samples Operator 設定オブジェクトの **samplesRegistry** をミラーリングされたレジストリーに設定します。
- 次に、Cluster Samples Operator を **Managed** に設定し、ミラーリングしたイメージストリームをインストールします。

5.13.2. 代替のレジストリーまたはミラーリングされたレジストリーでの Cluster Samples Operator イメージストリームの使用

Cluster Samples Operator によって管理される **openshift** namespace のほとんどのイメージストリームは、Red Hat レジストリーの registry.redhat.io にあるイメージを参照します。ミラーリングはこれらのイメージストリームには適用されません。



重要

jenkins、**jenkins-agent-maven**、および **jenkins-agent-nodejs** イメージストリームは、インストールペイロードからのもので、Samples Operator によって管理されるため、これらのイメージストリームには追加のミラーリングの手順は必要ありません。

Sample Operator 設定ファイルの **samplesRegistry** フィールドの registry.redhat.io への設定は、これはすでに Jenkins イメージおよびイメージストリーム以外のすべての registry.redhat.io に送信されているため不要になります。



注記

cli、**installer**、**must-gather**、および **tests** イメージストリームはインストールペイロードの一部ですが、Cluster Samples Operator によって管理されません。これらについては、この手順で扱いません。



重要

Cluster Samples Operator は、切断された環境で **Managed** に設定する必要があります。イメージストリームをインストールするには、ミラーリングされたレジストリーが必要です。

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。
- ミラーレジストリーのプルシークレットの作成。

手順

1. ミラーリングする特定のイメージストリームのイメージにアクセスします。

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep registry.redhat.io
```

2. ネットワークが制限された環境で必要とするイメージストリームに関連付けられた registry.redhat.io のイメージを定義されたミラーのいずれかにミラーリングします。

```
$ oc image mirror registry.redhat.io/rhsc/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhsc/ruby-25-rhel7:latest
```

3. 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
```

4. 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
```

5. Cluster Samples Operator 設定オブジェクトの **samplesRegistry** フィールドを、ミラー設定で定義されたミラーの場所の **hostname** の部分を含むように更新します。

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



注記

これは、イメージストリームのインポートプロセスでミラーまたは検索メカニズムが使用されないのが必要になります。

6. Cluster Samples Operator 設定オブジェクトの **skippedImagestreams** フィールドにミラーリングされないイメージストリームを追加します。または、サンプルイメージストリームのいずれもサポートする必要がない場合は、Cluster Samples Operator を Cluster Samples Operator 設定オブジェクトの **Removed** に設定します。



注記

Cluster Samples Operator は、イメージストリームのインポートに失敗した場合にアラートを発行しますが、Cluster Samples Operator は定期的に再試行する場合もあれば、それらを再試行していないように見える場合もあります。

openshift namespace のテンプレートの多くはイメージストリームを参照します。そのため、**Removed** を使用してイメージストリームとテンプレートの両方を除去すると、イメージストリームのいずれかが欠落しているためにテンプレートが正常に機能しない場合にテンプレートの使用を試行する可能性がなくなります。

5.13.3. 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

ポジトリリー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

- 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。

- 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
```

- 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
```

- 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)
```

5.14. CLUSTER SAMPLE OPERATOR イメージストリームタグの定期的なインポートの設定

新しいバージョンが利用可能になったときにイメージストリームタグを定期的にインポートすることで、Cluster Sample Operator イメージの最新バージョンに常にアクセスできるようになります。

手順

- 次のコマンドを実行して、**openshift** namespace のすべてのイメージストリームを取得します。

```
oc get imagestreams -n openshift
```

- 次のコマンドを実行して、**openshift** namespace のすべてのイメージストリームのタグを取得します。

```
$ oc get is <image-stream-name> -o jsonpath='{range .spec.tags[*]}{.name}{\t'}{.from.name}{\n'}{end}' -n openshift
```

以下に例を示します。

```
$ oc get is ubi8-openjdk-17 -o jsonpath='{range .spec.tags[*]}{.name}{\t'}{.from.name}{\n'}{end}' -n openshift
```

出力例

```
1.11 registry.access.redhat.com/ubi8/openjdk-17:1.11
1.12 registry.access.redhat.com/ubi8/openjdk-17:1.12
```

3. 次のコマンドを実行して、イメージストリームに存在する各タグのイメージの定期的なインポートをスケジュールします。

```
$ oc tag <repository/image> <image-stream-name:tag> --scheduled -nopenshift
```

以下に例を示します。

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.11 ubi8-openjdk-17:1.11 --scheduled -nopenshift
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.12 ubi8-openjdk-17:1.12 --scheduled -nopenshift
```

このコマンドにより、OpenShift Container Platform はこの特定のイメージストリームタグを定期的に更新します。この期間はクラスター全体のデフォルトで 15 分に設定されます。

4. 次のコマンドを実行して、定期的なインポートのスケジュールステータスを確認します。

```
oc get imagestream <image-stream-name> -o jsonpath="{range .spec.tags[*]}Tag: {.name}{\t}Scheduled: {.importPolicy.scheduled}{\n}{end}" -nopenshift
```

以下に例を示します。

```
oc get imagestream ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}Tag: {.name}{\t}Scheduled: {.importPolicy.scheduled}{\n}{end}" -nopenshift
```

出力例

```
Tag: 1.11 Scheduled: true
Tag: 1.12 Scheduled: true
```

第6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。

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

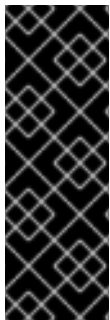
RHEL コンピュータノードについて理解し、これを使用します。

6.1.1. 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 4.10 では、**x86_64** アーキテクチャー上でユーザープロビジョニングまたはインストーラープロビジョニングのインフラストラクチャーインストールを使用する場合、クラスター内のコンピューティングマシンとして Red Hat Enterprise Linux (RHEL) マシンを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。

クラスターで RHEL コンピュータマシンを使用することを選択した場合は、すべてのオペレーティングシステムのライフサイクル管理とメンテナンスを担当します。システムの更新を実行し、パッチを適用し、その他すべての必要なタスクを完了する必要があります。

インストーラーがプロビジョニングしたインフラストラクチャークラスターの場合、インストーラーがプロビジョニングしたインフラストラクチャークラスターの自動スケーリングにより Red Hat Enterprise Linux CoreOS (RHCOS) コンピューティングマシンがデフォルトで追加されるため、RHEL コンピューティングマシンを手動で追加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。
- 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。

6.1.2. 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 % 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 8.4 から 8.7](#) (最小インストールオプション付き)。



重要

OpenShift Container Platform クラスターへの RHEL 7 コンピュータマシンの追加はサポートされません。

以前の OpenShift Container Platform のバージョンで以前にサポートされていた RHEL 7 コンピュータマシンがある場合、RHEL 8 にアップグレードすることはできません。新しい RHEL 8 ホストをデプロイする必要がある、古い RHEL 7 ホストを削除する必要があります。詳細は、ノードの管理セクションを参照してください。

OpenShift Container Platform で非推奨となったか、削除された主な機能の最新の一覧については、OpenShift Container Platform リリースノートの [非推奨および削除された機能セクション](#)を参照してください。

- 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。RHEL 8 ドキュメントの [Installing a RHEL 8 system with FIPS mode enabled](#)を参照してください。



重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピュータからインストールプログラムを実行する必要があります。RHEL での FIPS モードの設定の詳細は、[FIPS モードでのシステムのインストール](#)を参照してください。プロセス暗号化ライブラリーでの FIPS 検証済みまたはモジュールの使用は、**x86_64** アーキテクチャーでの OpenShift Container Platform デプロイメントでのみサポートされます。

- NetworkManager 1.0 以降。
- 1vCPU。
- 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 1GB のハードディスク領域。
- 一時ディレクトリーを含むファイルシステムの最小 1GB 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ーアクセス制御は、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。

あります。

関連情報

- [ノードの削除](#)

6.1.2.1. 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。

6.1.3. Playbook 実行のためのマシンの準備

Red Hat Enterprise Linux (RHEL) 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 4.10 クラスターに追加する前に、新たなノードをクラスターに追加する Ansible Playbook を実行する RHEL 7 または RHEL 8 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できる必要があります。

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** 権限を持つユーザーとしてログインしている。

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 RHEL 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 4.10 で必要なリポジトリを有効にします。

- RHEL 8 システムでは、次のコマンドを実行します。

```
# subscription-manager repos \
  --enable="rhel-8-for-x86_64-baseos-rpms" \
  --enable="rhel-8-for-x86_64-appstream-rpms" \
  --enable="ansible-2.9-for-rhel-8-x86_64-rpms" \
  --enable="rhocp-4.10-for-rhel-8-x86_64-rpms"
```

- RHEL 7 システムでは、次のコマンドを実行します。

```
# 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-4.10-rpms"
```



重要

OpenShift Container Platform 4.10.23 の時点で、RHEL 7 で Ansible Playbook を実行することは非推奨となり、既存のインストールを更新する目的でのみ推奨されています。OpenShift Container Platform 4.11 以降、Ansible Playbook は RHEL 8 に対してのみ提供されています。

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。

6.1.4. 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、必要なリ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 4.10 で必要なリポジトリのみを有効にします。

```
# subscription-manager repos \
  --enable="rhel-8-for-x86_64-baseos-rpms" \
  --enable="rhel-8-for-x86_64-appstream-rpms" \
  --enable="rhocp-4.10-for-rhel-8-x86_64-rpms" \
  --enable="fast-datapath-for-rhel-8-x86_64-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、ワーカー上の OpenShift Container Platform ログにはアクセスできません。

6.1.5. 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 4.10 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-rhel8-0.example.com
mycluster-rhel8-1.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 ❶
```

- ❶ **<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。

6.1.6. 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
ansible_user	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 root です。
ansible_become	ansible_user の値が root ではない場合、 ansible_become を True に設定する必要があります、 ansible_user 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。	True 。値が True 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。
openshift_kubeconfig_path	クラスターの kubeconfig 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。

6.1.7. 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ル対象外 (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-emptydir-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> 1
```

- 1 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。

6.2. 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。

6.2.1. 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。

6.2.2. 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タマシンの 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。

手順

1. ISO ファイルを使用して、追加のコンピュータマシンに 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. オプションを指定したり、ライブ起動シーケンスを中断したりせずに、RHCOS ISO イメージを起動します。インストーラーが RHCOS ライブ環境でシェルプロンプトを起動するのを待ちます。



注記

RHCOS インストールの起動プロセスを中断して、カーネル引数を追加できます。ただし、この ISO 手順では、カーネル引数を追加する代わりに、次の手順で概説するように **coreos-installer** コマンドを使用する必要があります。

3. **coreos-installer** コマンドを実行し、インストール要件を満たすオプションを指定します。少なくとも、ノードタイプの Ignition 設定ファイルを参照する URL と、インストール先のデバイスを指定する必要があります。

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> ❶ ❷
```

- ❶ コア ユーザーにはインストールを実行するために必要な root 権限がないため、**sudo** を使用して **coreos-installer** コマンドを実行する必要があります。
- ❷ **--ignition-hash** オプションは、Ignition 設定ファイルを HTTP URL を使用して取得し、クラスターノードの Ignition 設定ファイルの信頼性を検証するために必要です。**<digest>** は、先の手順で取得した Ignition 設定ファイル SHA512 ダイジェストです。



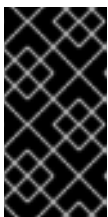
注記

TLS を使用する HTTPS サーバーを使用して Ignition 設定ファイルを提供する必要がある場合は、**coreos-installer** を実行する前に内部認証局 (CA) をシステム信頼ストアに追加できます。

以下の例では、**/dev/sda** デバイスへのブートストラップノードのインストールを初期化します。ブートストラップノードの Ignition 設定ファイルは、IP アドレス 192.168.1.2 で HTTP Web サーバーから取得されます。

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

4. マシンのコンソールで RHCOS インストールの進捗を監視します。



重要

OpenShift Container Platform のインストールを開始する前に、各ノードでインストールが成功していることを確認します。インストールプロセスを監視すると、発生する可能性のある RHCOS インストールの問題の原因を特定する上でも役立ちます。

5. 継続してクラスター用の追加のコンピュータマシンを作成します。

6.2.3. 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タマシンの 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>-live-kernel-<architecture> ❶
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img ❷

```

- ❶ HTTP サーバーにアップロードしたライブ **kernel** ファイルの場所を指定します。
- ❷ 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** パラメーターはライブ **initramfs** ファイルの場所であり、**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であり、**coreos.live.rootfs_url** パラメーター値はライブ **rootfs** ファイルの場所になります。**coreos.inst.ignition_url** および **coreos.live.rootfs_url** パラメーターは HTTP および HTTPS のみをサポートします。

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>-live-kernel-<architecture> initrd=main
coreos.inst.install_dev=/dev/sda coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.<architecture>.img

```

1

```
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.<architecture>.img
```

2

1

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=main** 引数は UEFI システムでの起動に必要であり、**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であり、**coreos.live.rootfs_url** パラメーター値は **rootfs** のライブファイルの場所です。**coreos.inst.ignition_url** および **coreos.live.rootfs_url** パラメーターは HTTP および HTTPS のみをサポートします。

2

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。

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。

1. 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トマシンを作成します。

6.2.4. 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、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	v1.23.0
master-1	Ready	master	63m	v1.23.0
master-2	Ready	master	64m	v1.23.0

出力には作成したすべてのマシンが一覧表示されます。



注記

上記の出力には、一部の CSR が承認されるまで、ワーカーノード (ワーカーノードとも呼ばれる) が含まれない場合があります。

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-
bootstrapper Pending
csr-8vnps  15m  system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper 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。

3. 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 が承認された後に、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、手動の承認が必要になります。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。

注記

ベアメタルおよび他の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などのマシン API ではないプラットフォームで実行されているクラスターの場合、kubelet 提供証明書要求 (CSR) を自動的に承認する方法を実装する必要があります。要求が承認されない場合、API サーバーが kubelet に接続する際に提供証明書が必須であるため、**oc exec**、**oc rsh**、および **oc logs** コマンドは正常に実行できません。Kubelet エンドポイントにアクセスする操作には、この証明書の承認が必要です。この方法は新規 CSR の有無を監視し、CSR が **system:node** または **system:admin** グループの **node-bootstrapper** サービスアカウントによって提出されていることを確認し、ノードのアイデンティティーを確認します。

- 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> 1
```

1 **<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
```



注記

一部の Operator は、一部の CSR が承認されるまで利用できない可能性があります。

4. 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
...
```

5. 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> 1
```

1 **<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
```

6. 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   v1.23.0
master-1  Ready    master   73m   v1.23.0
master-2  Ready    master   74m   v1.23.0
worker-0  Ready    worker   11m   v1.23.0
worker-1  Ready    worker   11m   v1.23.0
```


**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。

6.2.5. AWS でのカスタム `/var` パーティションを持つ新規 RHCOS ワーカーノードの追加

OpenShift Container Platform は、ブートストラップ時に処理されるマシン設定を使用したインストール時のデバイスのパーティション設定をサポートします。ただし、`/var` パーティション設定を使用する場合は、デバイス名はインストール時に決定する必要があり、変更することはできません。デバイス命名スキーマが異なる場合は、異なるインスタンスタイプをノードとして追加することはできません。たとえば、**m4.large** インスタンスにデフォルトの AWS デバイス名 (**dev/xvdb**) で `/var` パーティションを設定した場合、**m5.large** インスタンスはデフォルトで `/dev/nvme1n1` デバイスを使用するため、直接 AWS **m5.large** インスタンスを追加することはできません。異なる命名スキーマにより、デバイスはパーティション設定に失敗する可能性があります。

本セクションの手順では、インストール時に設定したものとは異なるデバイス名を使用するインスタンスと共に、新規の Red Hat Enterprise Linux CoreOS (RHCOS) コンピュータノードを追加する方法を説明します。カスタムユーザーデータシークレットを作成し、新規マシンセットを設定します。これらの手順は AWS クラスタに固有のものです。この原則は、他のクラウドデプロイメントにも適用されます。ただし、デバイスの命名スキーマは他のデプロイメントでは異なり、ケースごとに決定する必要があります。

手順

1. コマンドラインで、**openshift-machine-api** namespace に移動します。

```
$ oc project openshift-machine-api
```

2. **worker-user-data** シークレットから新規シークレットを作成します。

- a. シークレットの **userData** セクションをテキストファイルにエクスポートします。

```
$ oc get secret worker-user-data --template='{{index .data.userData | base64decode}}' |
jq > userData.txt
```

- b. テキストファイルを編集して、新規ノードに使用するパーティションの **storage**、**filesystems**、および **systemd** スタンザを追加します。必要に応じて [Ignition 設定パラメーター](#) を指定できます。

**注記**

ignition スタンザの値は変更しないでください。

```
{
  "ignition": {
    "config": {
      "merge": [
```

```

    {
      "source": "https:...."
    }
  ],
},
"security": {
  "tls": {
    "certificateAuthorities": [
      {
        "source": "data:text/plain;charset=utf-8;base64,.....=="
      }
    ]
  }
},
"version": "3.2.0"
},
"storage": {
  "disks": [
    {
      "device": "/dev/nvme1n1", ❶
      "partitions": [
        {
          "label": "var",
          "sizeMiB": 50000, ❷
          "startMiB": 0 ❸
        }
      ]
    }
  ],
  "filesystems": [
    {
      "device": "/dev/disk/by-partlabel/var", ❹
      "format": "xfs", ❺
      "path": "/var" ❻
    }
  ],
  "systemd": {
    "units": [ ❼
      {
        "contents": "[Unit]\nBefore=local-
fs.target\n[Mount]\nWhere=/var\nWhat=/dev/disk/by-
partlabel/var\nOptions=defaults,pquota\n[Install]\nWantedBy=local-fs.target\n",
        "enabled": true,
        "name": "var.mount"
      }
    ]
  }
}
}

```

- ❶ AWS ブロックデバイスへの絶対パスを指定します。
- ❷ データパーティションのサイズをメビバイト単位で指定します。
- ❸ メビバイト単位でパーティションの開始点を指定します。データパーティションを

- 4 `/var` パーティションへの絶対パスを指定します。
- 5 ファイルシステムのフォーマットを指定します。
- 6 Ignition がルートファイルシステムがマウントされる場所に対して相対的な場所で行われる、ファイルシステムのマウントポイントを指定します。これは実際のルートにマウントする場所と同じである必要はありませんが、同じにすることが推奨されます。
- 7 `/dev/disk/by-partlabel/var` デバイスを `/var` パーティションにマウントする `systemd` マウントユニットを定義します。

- c. **disableTemplating** セクションを **work-user-data** シークレットからテキストファイルに展開します。

```
$ oc get secret worker-user-data --template='{{index .data.disableTemplating | base64decode}}' | jq > disableTemplating.txt
```

- d. 2つのテキストファイルから新しいユーザーデータのシークレットファイルを作成します。このユーザーデータのシークレットは、**userData.txt** ファイルの追加のノードパーティション情報を新規作成されたノードに渡します。

```
$ oc create secret generic worker-user-data-x5 --from-file=userData=userData.txt --from-file=disableTemplating=disableTemplating.txt
```

3. 新規ノードの新規マシンセットを作成します。

- a. AWS 向けに設定される新規のマシンセット YAML ファイルを、以下のように作成します。必要なパーティションおよび新規に作成されたユーザーデータシークレットを追加します。

ヒント

既存のマシンセットをテンプレートとして使用し、新規ノード用に必要に応じてパラメーターを変更します。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: auto-52-92tf4
  name: worker-us-east-2-nvme1n1 1
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: auto-52-92tf4
      machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: auto-52-92tf4
        machine.openshift.io/cluster-api-machine-role: worker
```

```

machine.openshift.io/cluster-api-machine-type: worker
machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
spec:
  metadata: {}
  providerSpec:
    value:
      ami:
        id: ami-0c2dbd95931a
      apiVersion: awsproviderconfig.openshift.io/v1beta1
      blockDevices:
        - DeviceName: /dev/nvme1n1 ❷
          ebs:
            encrypted: true
            iops: 0
            volumeSize: 120
            volumeType: gp2
        - DeviceName: /dev/nvme1n2 ❸
          ebs:
            encrypted: true
            iops: 0
            volumeSize: 50
            volumeType: gp2
      credentialsSecret:
        name: aws-cloud-credentials
      deviceIndex: 0
      iamInstanceProfile:
        id: auto-52-92tf4-worker-profile
      instanceType: m6i.large
      kind: AWSMachineProviderConfig
      metadata:
        creationTimestamp: null
      placement:
        availabilityZone: us-east-2b
        region: us-east-2
      securityGroups:
        - filters:
            - name: tag:Name
              values:
                - auto-52-92tf4-worker-sg
      subnet:
        id: subnet-07a90e5db1
      tags:
        - name: kubernetes.io/cluster/auto-52-92tf4
          value: owned
      userDataSecret:
        name: worker-user-data-x5 ❹

```

- ❶ 新規ノードの名前を指定します。
- ❷ AWS ブロックデバイスへの絶対パスを指定します (ここでは暗号化された EBS ボリューム)。
- ❸ オプション: 追加の EBS ボリュームを指定します。
- ❹ ユーザーデータシークレットファイルを指定します。

- b. マシンセットを作成します。

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

マシンが利用可能になるまでに少し時間がかかる場合があります。

4. 新しいパーティションとノードが作成されたことを確認します。

- a. マシンセットが作成されていることを確認します。

```
$ oc get machineset
```

出力例

NAME	DESIRED	CURRENT	READY	AVAILABLE	AGE
ci-ln-2675bt2-76ef8-bdpsc-worker-us-east-1a	1	1	1	1	124m
ci-ln-2675bt2-76ef8-bdpsc-worker-us-east-1b	2	2	2	2	124m
worker-us-east-2-nvme1n1	1	1	1	1	2m35s 1

- 1** これは新規マシンセットです。

- b. 新規ノードが作成されていることを確認します。

```
$ oc get nodes
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-128-78.ec2.internal	Ready	worker	117m	v1.23.0+60f5a1c
ip-10-0-146-113.ec2.internal	Ready	master	127m	v1.23.0+60f5a1c
ip-10-0-153-35.ec2.internal	Ready	worker	118m	v1.23.0+60f5a1c
ip-10-0-176-58.ec2.internal	Ready	master	126m	v1.23.0+60f5a1c
ip-10-0-217-135.ec2.internal	Ready	worker	2m57s	v1.23.0+60f5a1c 1
ip-10-0-225-248.ec2.internal	Ready	master	127m	v1.23.0+60f5a1c
ip-10-0-245-59.ec2.internal	Ready	worker	116m	v1.23.0+60f5a1c

- 1** これは新しいノードです。

- c. カスタム **/var** パーティションが新しいノードに作成されていることを確認します。

```
$ oc debug node/<node-name> -- chroot /host lsblk
```

以下に例を示します。

```
$ oc debug node/ip-10-0-217-135.ec2.internal -- chroot /host lsblk
```

出力例

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
nvme0n1	202:0	0	120G	0	disk	
nvme0n1p1	202:1	0	1M	0	part	

```
| -nvme0n1p2 202:2 0 127M 0 part
| -nvme0n1p3 202:3 0 384M 0 part /boot
| `nvme0n1p4 202:4 0 119.5G 0 part /sysroot
| nvme1n1 202:16 0 50G 0 disk
| `nvme1n1p1 202:17 0 48.8G 0 part /var ❶
```

- ❶ **nvme1n1** デバイスが **/var** パーティションにマウントされます。

関連情報

- OpenShift Container Platform がディスクパーティションを使用する仕組みについては、[Disk partitioning](#) をしてください。

6.3. 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

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。ユーザーがプロビジョニングしたインフラストラクチャを持つクラスターでは、マシン API を使用するために追加の検証と設定が必要です。

インフラストラクチャプラットフォームタイプが **none** のクラスターは、マシン API を使用できません。この制限は、クラスターに接続されている計算マシンが、この機能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場合でも適用されます。このパラメーターは、インストール後に変更することはできません。

クラスターのプラットフォームタイプを表示するには、以下のコマンドを実行します。

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

6.3.1. マシンのヘルスチェック

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

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

注記

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

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

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

注記

タイムアウトについて注意深い検討が必要であり、ワークロードと要件を考慮してください。

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

チェックを停止するには、リソースを削除します。

6.3.1.1. マシンヘルスチェックのデプロイ時の制限

マシンヘルスチェックをデプロイする前に考慮すべき制限事項があります。

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

6.3.2. サンプル MachineHealthCheck リソース

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

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 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%" 7
nodeStartupTimeout: "10m" 8
```

- 1 デプロイするマシンヘルスチェックの名前を指定します。
- 2 3 チェックする必要のあるマシンプールのラベルを指定します。
- 4 追跡するマシンセットを `<cluster_name>-<label>-<zone>` 形式で指定します。たとえば、**prod-node-us-east-1a** とします。
- 5 6 ノードの状態のタイムアウト期間を指定します。タイムアウト期間の条件が満たされると、マシンは修正されます。タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。
- 7 ターゲットプールで同時に修復できるマシンの数を指定します。これはパーセンテージまたは整数として設定できます。正常でないマシンの数が **maxUnhealthy** で設定された制限を超える場合、修復は実行されません。
- 8 マシンが正常でないと判別される前に、ノードがクラスターに参加するまでマシンヘルスチェックが待機する必要のあるタイムアウト期間を指定します。



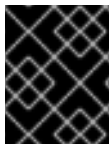
注記

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

6.3.2.1. 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。

適切な **maxUnhealthy** 値は、デプロイするクラスターの規模や、**MachineHealthCheck** が対応するマシンの数によって異なります。たとえば、**maxUnhealthy** 値を使用して複数のアベイラビリティゾーン間で複数のマシンセットに対応でき、ゾーン全体が失われると、**maxUnhealthy** の設定によりクラスター内で追加の修復を防ぐことができます。複数のアベイラビリティゾーンを持たないグローバル Azure リージョンでは、アベイラビリティセットを使用して高可用性を確保できます。

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

6.3.2.1.1. 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。

6.3.2.1.2. 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。

6.3.3. 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
```

6.3.4. 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. クラスター内にあるマシンを表示します。

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

3. 削除するマシンに注釈を設定します。

```
$ oc annotate machine/<machine_name> -n openshift-machine-api
machine.openshift.io/cluster-api-delete-machine="true"
```

4. 次のいずれかのコマンドを実行して、コンピュート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
```

ヒント

または、以下の YAML を適用してマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

コンピュート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。



重要

デフォルトでは、マシンコントローラーは、成功するまでマシンによってサポートされるノードをドレイン (解放) しようとします。Pod 中断バジエットの設定が間違っているなど、状況によっては、ドレイン操作が成功しない可能性があります。排水操作が失敗した場合、マシンコントローラーはマシンの取り外しを続行できません。

特定のマシンの **machine.openshift.io/exclude-node-draining** にアノテーションを付けると、ノードのドレイン (解放) を省略できます。

検証

- 目的のマシンの削除を確認します。

```
$ oc get machines
```

6.3.5. 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。

6.4. 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。



注記

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

podSPerCore は、ノードのプロセッサ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 **podSPerCore** が **10** に設定される場合、このノードで許可される Pod の最大数は **40** になります。

```
kubeletConfig:
  podSPerCore: 10
```

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

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

```
kubeletConfig:
  maxPods: 250
```

6.4.1. 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) を使用して kubelet パラメーターを編集できます。



注記

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

以下のガイダンスを参照してください。

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

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



注記

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

KubeletConfig CR の例

```
$ oc get kubeletconfig
```

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

KubeletConfig マシン設定を示す例

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

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

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

前提条件

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

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

以下に例を示します。

```
$ oc describe machineconfigpool worker
```

出力例

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

- ① ラベルが追加されると、**labels** の下に表示されます。

- b. ラベルが存在しない場合は、キー/値のペアを追加します。

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

手順

1. これは、選択可能なマシン設定オブジェクトを表示します。

—

```
$ oc get machineconfig
```

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

2. ノードあたりの最大 Pod の現在の値を確認します。

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

以下に例を示します。

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

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

出力例

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

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

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

❶ Machine Config Pool からラベルを入力します。

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

注記

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

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

- a. ラベルを使用してワーカーのマシン設定プールを更新します。

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

- b. **KubeletConfig** オブジェクトを作成します。

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

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

```
$ oc get kubeletconfig
```

出力例

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

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

4. 変更がノードに適用されていることを確認します。

- a. **maxPods** 値が変更されたワーカーノードで確認します。

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

- b. **Allocatable** スタンザを見つけます。

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

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

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

5. KubeletConfig オブジェクトの変更を確認します。

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

これは、以下の例のように **True** および **type:Success** のステータスを表示します。

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

6.4.2. 利用不可のワーカーノードの数の変更

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

手順

1. **worker** マシン設定プールを編集します。

```
$ oc edit machineconfigpool worker
```

2. **maxUnavailable** を必要な値に設定します。

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



重要

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

6.4.3. コントロールプレーンノードのサイジング

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

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

ワーカーノードの数	クラスター密度 (namespace)	CPU コア数	メモリー (GB)
24	500	4	16
120	1000	8	32
252	4000	16	64
501	4000	16	96

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



重要

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

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

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

 重要

以下の設定でのみ、実行中の OpenShift Container Platform 4.10 クラスターでコントロールプレーンのノードサイズを変更できます。

- ユーザーがプロビジョニングしたインストール方法でインストールされたクラスター。
- インストーラーによってプロビジョニングされたインフラストラクチャーインストール方法でインストールされた AWS クラスター。

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

 重要

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

 注記

OpenShift Container Platform 4.10 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。

6.4.4. 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-xx
  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。これはデフォルトポリシーになります。
- **static** このポリシーは、整数の CPU 要求を持つ保証された Pod 内のコンテナを許可します。また、ノードの排他的 CPU へのアクセスも制限します。**static** の場合は、小文字の **s** を使用する必要があります。

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 **cpuManagerPolicy** は、**KubeletConfig** CR の作成時に定義されます。
- ❷ **cpuManagerReconcilePeriod** は、**KubeletConfig** CR の作成時に定義されます。

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

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

出力例

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  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) 層 **Guaranteed** の Pod は、 **kubepods.slice** に配置されます。他の QoS 層の Pod は、 **kubepods** の子である **cgroups** に配置されます。

```

# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done

```

出力例

```

cpuset.cpus 1
tasks 32706

```

12. 対象のタスクで許可される CPU 一覧を確認します。

```

# grep ^Cpus_allowed_list /proc/32706/status

```

出力例

```

Cpus_allowed_list:      1

```

13. システム上の別の Pod (この場合は **burstable** QoS 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-burstable.slice/kubepods-burstable-
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

6.5. HUGE PAGE

Huge Page について理解し、これを設定します。

6.5.1. 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はしますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

6.5.2. 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。

6.5.3. Huge Page の設定

ノードは、OpenShift Container Platform クラスタで使用される 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。

6.5.3.1. 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** オブジェクトの作成

```
$ oc create -f hugepages-tuned-boottime.yaml
```

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

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

5. マシン設定プールを作成します。

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

断片化されていないメモリが十分にある場合、**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
```

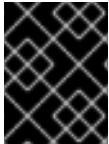


警告

TuneD ブートローダープラグインは現在、Red Hat Enterprise Linux CoreOS (RHCOS) 8.x ワーカーノードでサポートされています。Red Hat Enterprise Linux (RHEL) 7.x ワーカーノードの場合、TuneD ブートローダープラグインは現時点でサポートされていません。

6.6. 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

- [Nvidia GPU device plugin for COS-based operating system](#)
- [Nvidia official GPU device plugin](#)
- [Solarflare device plugin](#)
- [KubeVirt device plugins: vfio and kvm](#)
- [Kubernetes device plugin for IBM Crypto Express \(CEX\) cards](#)



注記

デバイスプラグイン参照の実装を容易にするため
に、[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。

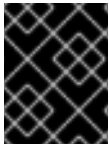
6.6.1. デバイスプラグインのデプロイ方法

- デモンセットは、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ー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。

6.6.2. 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用し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** リモートプロシーチャーコール (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。

6.6.3. 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用して公開するメカニズムを提供します。

1. 次のコマンドを入力して、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 ❶
```

- ❶ 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 ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr ❷
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true ❸
```

- ❶ CR に名前を割り当てます。
- ❷ Machine Config Pool からラベルを入力します。
- ❸ **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。

6.7. 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。

6.7.1. 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

```
apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Pod 仕様での容認の例

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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6.1 テイントおよび容認コンポーネント

パラメーター	説明						
key	key 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。						
value	value 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。						
effect	effect は以下のいずれかにすることができます。 <table><tr><td>NoSchedule ^[1]</td><td> • 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。 • ノードの既存 Pod はそのままになります。 </td></tr><tr><td>PreferNoSchedule</td><td> • 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。 • ノードの既存 Pod はそのままになります。 </td></tr><tr><td>NoExecute</td><td> • 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。 • 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。 </td></tr></table>	NoSchedule ^[1]	• 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。 • ノードの既存 Pod はそのままになります。	PreferNoSchedule	• 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。 • ノードの既存 Pod はそのままになります。	NoExecute	• 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。 • 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。
NoSchedule ^[1]	• 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。 • ノードの既存 Pod はそのままになります。						
PreferNoSchedule	• 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。 • ノードの既存 Pod はそのままになります。						
NoExecute	• 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。 • 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。						
operator	<table><tr><td>Equal</td><td>key/value/effect 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>Exists</td><td>key/effect 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 value パラメーターを空のままにする必要があります。</td></tr></table>	Equal	key/value/effect 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。	Exists	key/effect 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 value パラメーターを空のままにする必要があります。		
Equal	key/value/effect 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。						
Exists	key/effect 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 value パラメーターを空のままにする必要があります。						

1. **NoSchedule** テイントをコントロールプレーン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。
以下に例を示します。

```
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
    name: my-node
  #...
  spec:
    taints:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
    #...

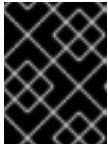
```

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**: ノードは準備状態にありません。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/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用して起動すると、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。
- **node.kubernetes.io/pid-pressure**: ノードが pid 不足の状態です。これはノード条件 **PIDPressure=True** に対応します。

**重要**

OpenShift Container Platform では、デフォルトの `pid.available` **evictionHard** は設定されません。

6.7.1.1. 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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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。

6.7.1.2. 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。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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。

6.7.1.3. 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/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。



注記

コントロールプレーンは、QoS クラスを持つ Pod に **`node.kubernetes.io/memory-pressure`** 容認も追加します。これは、Kubernetes が **Guaranteed** または **Burstable** QoS クラスで Pod を管理するためです。新しい **BestEffort** Pod は、影響を受けるノードにスケジュールされません。

6.7.1.4. 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です)。

`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 エビクションを防ぎます。

デフォルトでは、特定のゾーン内のノードの 55% 以上が異常である場合、ノードライフサイクルコントローラーはそのゾーンの状態を **PartialDisruption** に変更し、Pod の削除率が低下します。この状態の小さなクラスター (デフォルトでは 50 ノード以下) の場合、このゾーンのノードは汚染されず、排除が停止されます。

詳細については、Kubernetes ドキュメントの [Rate limits on eviction](#) を参照してください。

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。

```
apiVersion: v1
kind: Pod
metadata:
```

```

name: my-pod
#...
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300 ❶
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
#...

```

- ❶ 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 のエビクションを回避できます。

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。

6.7.1.5. 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

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - operator: "Exists"
#...

```

6.7.2. 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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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** を持つティントを **node1** に配置します。

注記

NoSchedule ティントをコントロールプレーン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** ティントが必要です。

以下に例を示します。

```
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
    name: my-node
#...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
#...
```

Pod の容認はノードのティントに一致します。いずれかの容認のある Pod は **node1** にスケジューリングされます。

6.7.3. マシンセットを使用したティントおよび容認の追加

マシンセットを使用してティ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ィントで更新されます。容認は、ノードに直接追加されたティントと同様に、マシンセットによって追加されるティントに応答します。

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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
```

```
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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  #...
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、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。

マシンセット仕様のテイントの例

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: my-machineset
  #...
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
```

ヒント

または、以下の YAML を適用してマシンセットをスケーリングすることもできます。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

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
```

または、以下を実行します。

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

マシンが起動するまで待ちます。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。

6.7.4. テイントおよび容認 (Toleration) 使用し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
```

ヒント

または、以下の YAML を適用してテイントを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: 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。

6.7.5. テイントおよび容認 (Toleration) を使用し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。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
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
```

ヒント

または、以下の YAML を適用してテイントを追加できます。

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
#...
```

6.7.6. 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
```

2. 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。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
```

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

6.8. TOPOLOGY MANAGER

Topology Manager について理解し、これを使用します。

6.8.1. 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。

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

none ポリシー

これはデフォルトのポリシーで、トポロジーの配置は実行しません。

best-effort ポリシー

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

6.8.2. Topology Manager のセットアップ

Topology Manager を使用するには、**cpumanager-enabled** という名前の **KubeletConfig** カスタムリソース (CR) で割り当てポリシーを設定する必要があります。CPU マネージャーをセットアップしている場合は、このファイルが存在している可能性があります。ファイルが存在しない場合は、作成できます。

前提条件

- CPU マネージャーのポリシーを **static** に設定します。

手順

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

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

```
$ oc edit KubeletConfig cpumanager-enabled

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static ❶
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node ❷
```

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

6.8.3. 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 はヒントプロバイダー (CPU マネージャーおよびデバイスマネージャー) を参照して、Pod のトポロジーヒントを取得します。

Topology Manager はこの情報を使用して、このコンテナに最適なトポロジーを保管します。この Pod の場合、CPU マネージャーおよびデバイスマネージャーは、リソース割り当ての段階でこの保存された情報を使用します。

6.9. 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もっと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。

6.10. 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:
  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、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、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、**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。

6.10.1. 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、**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。

1. OpenShift Container Platform Web コンソールで、**Home → Projects** に移動します。
  - a. **Create Project** をクリックします。
  - b. **clusterresourceoverride-operator** をプロジェクトの名前として指定します。
  - c. **Create** をクリックします。
2. **Operators → OperatorHub** に移動します。
  - a. 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。
  - b. **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。
  - c. **clusterresourceoverride-operator** が **Installed Namespace** について選択されていることを確認します。
  - d. **Update Channel** および **Approval Strategy** を選択します。
  - e. **Install** をクリックします。
3. **Installed Operators** ページで、**ClusterResourceOverride** をクリックします。
  - a. **ClusterResourceOverride Operator** 詳細ページで、**Create ClusterResourceOverride** をクリックします。
  - b. **Create ClusterResourceOverride** ページで、**YAML view** をクリックして、YAML テンプレートを編集し、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 ❶
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 ❷
```

```
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. **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:
    kubectrl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","met
adata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":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
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

mutatingWebhookConfigurationRef: 1
apiVersion: admissionregistration.k8s.io/v1
```



```
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。

6.10.2. 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、**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: "4.10"
  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
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. **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,"limitCPUMemoryPercent":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/v1
  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。

6.10.3. 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、**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:
  podResourceOverride:
    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

は、これを指定します。この値は、ノードのスケジューラによって、ノードのスケジューラに等しくなります。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。

2. 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が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" 1

# ...
```

- 1 このラベルを各プロジェクトに追加します。

6.11. 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。

6.11.1. 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。

6.11.1.1. 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。

6.11.1.2. 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。

6.11.2. 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タ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。

Pod は、優先度の高い順に 3 つの QoS クラスの 1 つとして指定されます。

表6.2 QoS (Quality of Service) クラス

優先順位	クラス名	説明
1(最高)	Guaranteed	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、Pod は Guaranteed として分類されます。
2	Burstable	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、Pod は Burstable として分類されます。
3(最低)	BestEffort	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、Pod は BestEffort 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、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。

6.11.2.1. 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。

6.11.3. 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。

6.11.4. 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 = 0
#...
```

```
$ 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

6.11.5. 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

- 次のコマンドを入力して、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。

```
$ oc edit machineconfigpool <name>
```

以下に例を示します。

```
$ oc edit machineconfigpool worker
```

出力例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
```



```

metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" ❶
  name: worker

```

- ❶ Labels の下にラベルが表示されます。

ヒント

ラベルが存在しない場合は、次のようなキー/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:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    cpuCfsQuota: ❸
      - "true"

```

- ❶ CR に名前を割り当てます。
- ❷ マシン設定プールからラベルを指定します。
- ❸ **cpuCfsQuota** パラメーターを **true** に設定します。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f <file_name>.yaml
```

6.11.6. 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が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。

6.11.7. 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
```

6.12. 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。

6.12.1. 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. namespace オブジェクトを編集して、次のアノテーションを追加します。

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    quota.openshift.io/cluster-resource-override-enabled: "false" 1
# ...
```

- 1 このアノテーションを **false** に設定すると、この namespace のオーバーコミットが無効になります。

6.13. 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。

6.13.1. 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、終了したコンテナを削除します。

エビクションしきい値がガー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。

以下の表は、エビクションしきい値の一覧です。

表6.3 コンテナのガーベージコレクションを設定するための変数

ノードの状態	エビクションシグナル	説明
MemoryPressure	memory.available	ノードで利用可能なメモリー。
DiskPressure	● nodefs.available ● nodefs.inodesFree ● imagefs.available ● imagefs.inodesFree	ノードのルートファイルシステム (nodefs) またはイメージファイルシステム (imagefs) で利用可能なディスク領域または i ノード。



注記

evictionHard の場合、これらのパラメーターをすべて指定する必要があります。すべてのパラメーターを指定しないと、指定したパラメーターのみが適用され、ガーベージコレクションが正しく機能しません。

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。

6.13.2. イメージがガー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

イメージガーベージコレクションは、実行中の Pod によって参照されていないイメージを削除します。

OpenShift Container Platform は、**cAdvisor** によって報告された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。

イメージのガー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6.4 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
imageMinimumGcAge	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。
imageGCHighThresholdPercent	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。
imageGCLowThresholdPercent	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、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。

6.13.3. 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

コンテナのガベージコレクションは終了したコンテナを削除します。イメージガベージコレクションは、実行中の Pod によって参照されていないイメージを削除します。

前提条件

1. 次のコマンドを入力して、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。

```
$ oc edit machineconfigpool <name>
```

以下に例を示します。

```
$ oc edit machineconfigpool worker
```

出力例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" ❶
  name: worker
#...
```

- ❶ Labels の下にラベルが表示されます。

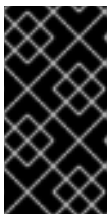
ヒント

ラベルが存在しない場合は、次のようなキー/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。



重要

ファイルシステムが1つの場合、または **/var/lib/kubelet** と **/var/lib/containers/** が同じファイルシステムにある場合、最も大きな値の設定が満たされるとエビクションがトリガーされます。ファイルシステムはエビクションをトリガーします。

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:
      pools.operator.machineconfiguration.openshift.io/worker: "" 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: ❻
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s ❼
    imageMinimumGCAge: 5m ❽
    imageGCHighThresholdPercent: 80 ❾
    imageGCLowThresholdPercent: 75 ❿
#...
```

- ❶ オブジェクトの名前。
- ❷ マシン設定プールからラベルを指定します。
- ❸ コンテナのガベージコレクションの場合: エビクションのタイプ: **evictionSoft** または **evictionHard**。
- ❹ コンテナのガベージコレクションの場合: 特定のエビクショントリガー信号に基づくエビクションしきい値。
- ❺ コンテナのガベージコレクションの場合: 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 コンテナのガベージコレクションの場合: 特定のエビクショントリガー信号に基づくエビクションしきい値。**evictionHard** の場合、これらのパラメーターをすべて指定する必要があります。すべてのパラメーターを指定しないと、指定したパラメーターのみが適用され、ガベージコレクションが正しく機能しません。
- ❼ コンテナのガベージコレクションの場合: エビクションプレッシャー状態から移行するまでの待機時間。
- ❽ イメージのガベージコレクションの場合: イメージが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最小保存期間。
- ❾ イメージガベージコレクションの場合: イメージガベージコレクションをトリガーするディスク使用率 (整数で表されます)。

- 10 イメージガベージコレクションの場合: イメージ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率 (整数で表されます)。

2. 以下のコマンドを実行して CR 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
```

検証

1. 次のコマンドを入力して、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' と表示されます。

```
$ oc get machineconfigpool
```

出力例

NAME	CONFIG	UPDATED	UPDATING
master	rendered-master-546383f80705bd5aeaba93	True	False
worker	rendered-worker-b4c51bb33ccaae6fc4a6a5	False	True

6.14. 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。

6.14.1. 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。

6.14.2. 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 デモンの適切なオブジェクトは更新されます。

管理状態

Operator 管理の状態は、デフォルトの Tuned CR を調整して設定されます。デフォルトで、Operator は Managed 状態であり、**spec.managementState** フィールドはデフォルトの Tuned CR に表示されません。Operator Management 状態の有効な値は以下のとおりです。

- Managed: Operator は設定リソースが更新されるとそのオペランドを更新します。
- Unmanaged: Operator は設定リソースへの変更を無視します。
- Removed: Operator は Operator がプロビジョニングしたオペランドおよびリソースを削除します。

プロファイルデータ

profile: セクションは、TuneD プロファイルおよびそれらの名前を一覧表示します。

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

推奨プロファイル

profile: 選択ロジックは、CR の **recommend:** セクションによって定義されます。 **recommend:** セクションは、選択基準に基づくプロファイルの推奨項目の一覧です。

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

一覧の個別項目:

```
- machineConfigLabels: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
  operand: ❼
  debug: <bool> ❽
```

- 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、**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)。

- 6 一致に適用する TuneD プロファイル。例: **tuned_profile_1**
- 7 オプションのオペランド設定。
- 8 TuneD デーモンのデバッグオンまたはオフを有効にします。オプションは、オンの場合は **true**、オフの場合は **false** です。デフォルトは **false** です。

<match> は、以下のように再帰的に定義されるオプションの一覧です。

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

- 1 ノードまたは Pod のラベル名。
- 2 オプションのノードまたは Pod のラベルの値。省略されている場合も、**<label_name>** があるだけで一致条件を満たします。
- 3 オプションのオブジェクトタイプ (**node** または **pod**)。省略されている場合は、**node** が想定されます。
- 4 オプションの **<match>** 一覧。

<match> が省略されない場合、ネストされたすべての **<match>** セクションが **true** に評価される必要もあります。そうでない場合には **false** が想定され、それぞれの **<match>** セクションのあるプロファイルは適用されず、推奨されません。そのため、ネスト化 (子の **<match>** セクション) は論理 AND 演算子として機能します。これとは逆に、**<match>** 一覧のいずれかの項目が一致する場合は、**<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は、マシン設定プールベースのマッチングが指定の **recommend:** 一覧の項目に対してオンになります。**<mcLabels>** はマシン設定のラベルを指定します。マシン設定は、プロファイル **<tuned_profile_name>** についてカーネル起動パラメーターなどのホスト設定を適用するために自動的に作成されます。この場合は、マシン設定セレクターが **<mcLabels>** に一致するすべてのマシン設定プールを検索し、プロファイル **<tuned_profile_name>** を確認されるマシン設定プールが割り当てられるすべてのノードに設定する必要があります。マスターロールとワーカーのロールの両方を持つノードをターゲットにするには、マスターロールを使用する必要があります。

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

例: ノード/Pod ラベルベースのマッチング

```

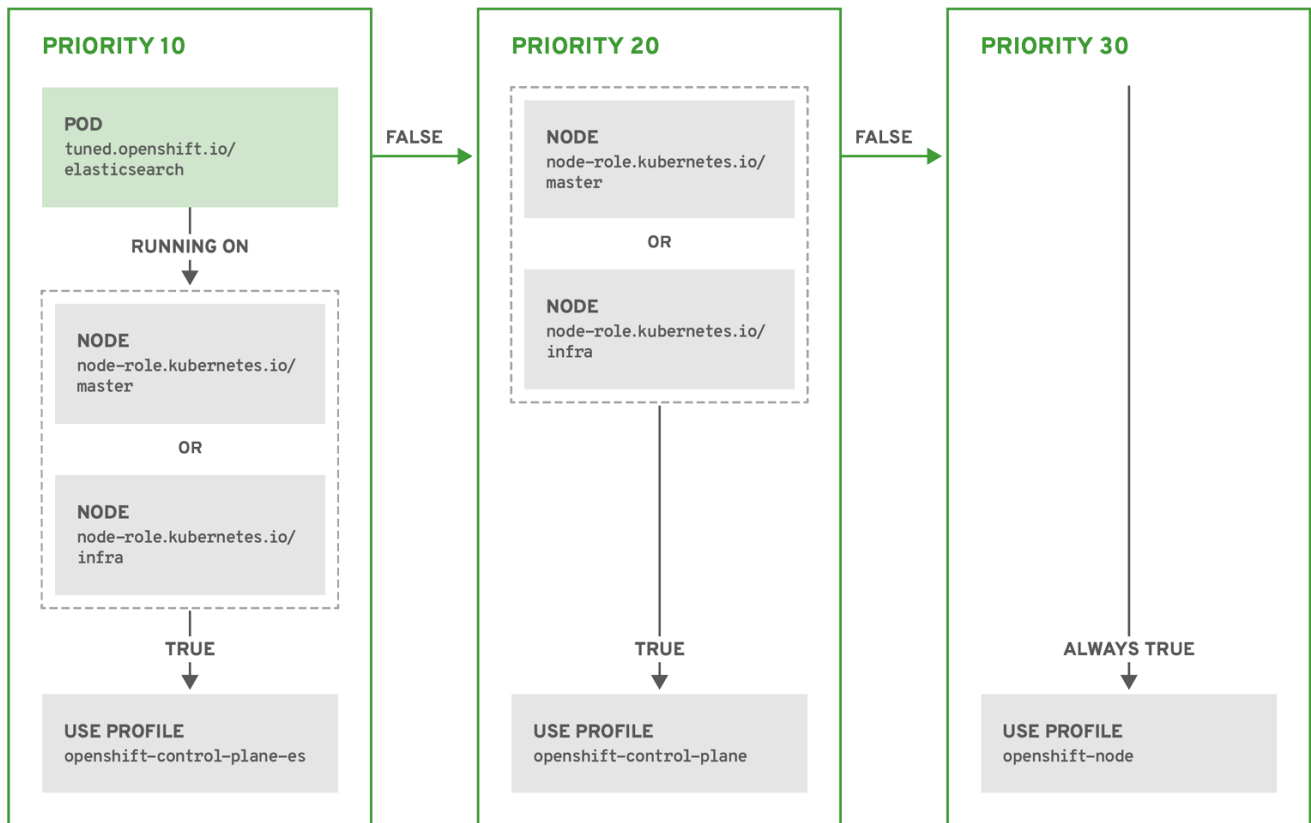
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  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。これがない場合は、**<match>** セクション全体が **false** として評価されます。このラベルを持つこのような Pod がある場合に、**<match>** セクションが **true** に評価されるようにするには、ノードラベルを **node-role.kubernetes.io/master** または **node-role.kubernetes.io/infra** にする必要もあります。

優先順位が **10** のプロファイルのラベルが一致した場合は、**openshift-control-plane-es** プロファイルが適用され、その他のプロファイルは考慮されません。ノード/Pod ラベルの組み合わせが一致しない場合は、2 番目に高い優先順位プロファイル (**openshift-control-plane**) が考慮されます。このプロファイルは、コンテナ化された TuneD Pod が **node-role.kubernetes.io/master** または **node-role.kubernetes.io/infra** ラベルを持つノードで実行される場合に適用されます。

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



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。

6.14.3. クラスタ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:
  recommend:
  - profile: "openshift-control-plane"
    priority: 30
    match:
    - label: "node-role.kubernetes.io/master"
    - label: "node-role.kubernetes.io/infra"

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

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

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

6.14.4. 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
- bootloader

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

- script
- systemd



警告

TuneD ブートローダープラグインは現在、Red Hat Enterprise Linux CoreOS (RHCOS) 8.x ワーカーノードでサポートされています。Red Hat Enterprise Linux (RHEL) 7.x ワーカーノードの場合、TuneD ブートローダープラグインは現時点でサポートされていません。

その他の参考資料

- [利用可能な TuneD プラグイン](#)
- [TuneD を使い始める](#)

6.15. 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. 次のコマンドを入力して、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。

```
$ oc edit machineconfigpool <name>
```

以下に例を示します。

```
$ oc edit machineconfigpool worker
```

出力例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
```

```

metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" ❶
  name: worker
#...

```

- ❶ Labels の下にラベルが表示されます。

ヒント

ラベルが存在しない場合は、次のようなキー/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。

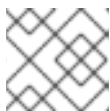
max-pods CR の設定例

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    podsPerCore: 10 ❸
    maxPods: 250 ❹
#...

```

- ❶ CR に名前を割り当てます。
- ❷ マシン設定プールから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. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f <file_name>.yaml
```

検証

1. 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7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。

7.1. 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** という名前のカスタムリソース (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 のフィールドを指定します。

CNO 設定は、**Network.config.openshift.io** API グループの **Network** API からクラスターのインストール時に以下のフィールドを継承し、これらのフィールドは変更できません。

clusterNetwork

Pod IP アドレスの割り当てに使用する IP アドレスプール。

serviceNetwork

サービスの IP アドレスプール。

defaultNetwork.type

OpenShift SDN または OVN-Kubernetes などのクラスターネットワークプロバイダー。



注記

クラスターのインストール後に、直前のセクションで一覧表示されているフィールドを変更することはできません。

7.2. クラスター全体のプロキシの有効化

Proxy 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、**Proxy** 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 Proxy** 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 **Proxy** オブジェクトのみがサポートされ、追加のプロキシを作成することはできません。

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 config map 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** **Proxy** オブジェクトから参照される config map 名。
- 4** config map 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 **Proxy** 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: https://<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。URL スキームは **http** または **https** である必要があります。URL スキームをサポートするプロキシの URL を指定します。たとえば、ほとんどのプロキシは、**https** を使用するように設定されていても、**http** しかサポートしていない場合、エラーを報告します。このエラーメッセージはログに反映されず、代わりにネットワーク接続エラーのように見える場合があります。クラスターからの **https** 接続をリッスンするプロキシを使用している場合は、プロキシが使用する CA と証明書を受け入れるようにクラスターを設定する必要がある場合があります。
- 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 config map の参照。ここで参照する前に config map が存在する必要があります。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。

7.3. 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: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: 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。

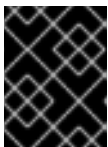
2. **DNS** カスタムリソースにパッチを適用して、パブリックゾーンを削除します。

```

$ 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: <infrastructure_id>-int
kubernetes.io/cluster/<infrastructure_id>-wfp4: owned
status: {}
```

7.4. 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用し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 アドレスを使用し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 アドレスを使用した非標準ポートへのトラフィックを許可します。
NodePort を設定する	クラスターのすべてのノードでサービスを公開します。

7.5. ノードポートサービス範囲の設定

クラスター管理者は、利用可能なノードのポート範囲を拡張できます。クラスターで多数のノードポートが使用される場合、利用可能なポートの数を増やす必要がある場合があります。

デフォルトのポート範囲は **30000-32767** です。最初にデフォルト範囲を超えて拡張した場合でも、ポート範囲を縮小することはできません。

7.5.1. 前提条件

- クラスターインフラストラクチャーは、拡張された範囲内で指定するポートへのアクセスを許可する必要があります。たとえば、ノードのポート範囲を **30000-32900** に拡張する場合、ファイアウォールまたはパケットフィルタリングの設定によりこれに含まれるポート範囲 **32768-32900** を許可する必要があります。

7.5.1.1. ノードのポート範囲の拡張

クラスターのノードポート範囲を拡張できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

1. ノードのポート範囲を拡張するには、以下のコマンドを入力します。**<port>** を、新規の範囲内で最大のポート番号に置き換えます。

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
  '{
    "spec":
      { "serviceNodePortRange": "30000-<port>" }
  }'
```

ヒント

または、以下の YAML を適用してノードのポート範囲を更新することもできます。

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

出力例

```
network.config.openshift.io/cluster patched
```

2. 設定がアクティブであることを確認するには、以下のコマンドを入力します。更新が適用されるまでに数分の時間がかかることがあります。

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo "service-node-port-range": "[[:digit:]]+-[[:digit:]]+"
```

出力例

```
"service-node-port-range":["30000-33000"]
```

7.6. ネットワークポリシーの設定

クラスター管理者またはプロジェクト管理者として、プロジェクトのネットワークポリシーを設定できます。

7.6.1. 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 4.10 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。

OpenShift SDN クラスターネットワークプロバイダーは、**egress** フィールドで指定された egress ネットワークポリシーをサポートするようになりました。



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。ただし、ホストネットワークの Pod に接続する Pod はネットワークポリシールールの影響を受ける可能性があります。

ネットワークポリシーは、ローカルホストまたは常駐ノードからのトラフィックをブロックすることはできません。

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。

ネットワークポリシーは、TCP、UDP、および SCTP プロトコルにのみ適用されます。他のプロトコルは影響を受けません。

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。

```
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

```

- 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て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。

7.6.2. 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 ❶
spec:
  podSelector: ❷
    matchLabels:
      app: mongodb
  ingress:
    - from:
        - podSelector: ❸
            matchLabels:
              app: app
      ports: ❹
        - protocol: TCP
          port: 27017

```

- ❶ NetworkPolicy オブジェクトの名前。
- ❷ ポリシーが適用される Pod を説明するセレクター。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。
- ❸ ポリシーオブジェクトが入力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、NetworkPolicy と同じ namespace にある Pod を照合して検索します。
- ❹ トラフィックを受け入れる1つ以上の宛先ポートのリスト。

7.6.3. 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、クラスター内の任意の 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.networking.k8s.io/default-deny created
```



注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接作成できます。

7.6.4. 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:

```
$ cat << EOF | oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
```

```

ingress:
- from:
  - namespaceSelector:
      matchLabels:
        policy-group.network.openshift.io/ingress: ""
podSelector: {}
policyTypes:
- Ingress
EOF

```



注記

policy-group.network.openshift.io/ingress: ""は、OpenShift SDN の推奨の namespace セレクターラベルです。**network.openshift.io/policy-group: ingress** namespace セレクターラベルを使用できますが、これはレガシーラベルです。

- 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

```

- d. **allow-from-kube-apiserver-operator** 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-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
    policyTypes:
    - Ingress
EOF

```

詳細は、新規の [New kube-apiserver-operator webhook controller validating health of webhook](#) を参照してください。

2. オプション: 以下のコマンドを実行し、ネットワークポリシー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 describe networkpolicy
```

出力例

```

Name:      allow-from-openshift-ingress
Namespace: example1
Created on: 2020-06-09 00:28:17 -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:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress

```

```

Name:      allow-from-openshift-monitoring
Namespace: example1
Created on: 2020-06-09 00:29:57 -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:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress

```

7.6.5. 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。

7.6.6. 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、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. **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。

7.6.6.1. 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。
- 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。
- 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。

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
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                kubernetes.io/metadata.name: openshift-kube-apiserver-operator
            podSelector:
              matchLabels:
                app: kube-apiserver-operator
          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

```

7.7. サポートされる構成

以下の設定は、Red Hat OpenShift Service Mesh の現行リリースでサポートされます。

7.7.1. サポート対象プラットフォーム

Red Hat OpenShift Service Mesh Operator は、複数のバージョンの **ServiceMeshControlPlane** リソースをサポートします。バージョン 2.4 の Service Mesh コントロールプレーンは、以下のプラットフォームバージョンでサポートされます。

- Red Hat OpenShift Container Platform バージョン 4.10 以降
- Red Hat OpenShift Dedicated バージョン 4

- Azure Red Hat OpenShift (ARO) バージョン 4
- Red Hat OpenShift Service on AWS (ROSA)

7.7.2. サポートされない設定

明示的にサポート対象外とされているケースには、以下が含まれます。

- OpenShift Online は Red Hat OpenShift Service Mesh に対してはサポートされていません。
- Red Hat OpenShift Service Mesh では、Service Mesh が実行されているクラスター以外にあるマイクロサービスの管理はサポートしていません。

7.7.3. サポートされるネットワーク設定

Red Hat OpenShift Service Mesh は以下のネットワーク設定をサポートします。

- OpenShift-SDN
- OVN-Kubernetes は、サポートされているすべてのバージョンの OpenShift Container Platform で使用できます。
- OpenShift Container Platform で認定され、さらに Service Mesh 適合テストに合格したサードパーティーの Container Network Interface(CNI) プラグイン。詳細は、[認定 OpenShift CNI プラグイン](#) を参照してください。

7.7.4. Service Mesh でサポートされる設定

- Red Hat OpenShift Service Mesh の本リリースは、OpenShift Container Platform x86_64、IBM Z、および IBM Power でのみ利用できます。
  - IBM Z は OpenShift Container Platform 4.10 以降でのみサポートされます。
  - IBM Power は OpenShift Container Platform 4.10 以降でのみサポートされます。
- すべての Service Mesh コンポーネントが単一の OpenShift Container Platform クラスター内に含まれる設定。
- 仮想マシンなどの外部サービスを統合しない設定。
- Red Hat OpenShift Service Mesh は、明示的に文書化されている場合を除き、**EnvoyFilter** の設定はサポートしていません。

7.7.5. Kiali のサポートされる設定

- Kiali コンソールは、Google Chrome、Microsoft Edge、Mozilla Firefox、または Apple Safari ブラウザーの最新の 2 つのリリースでのみサポートされています。
- **OpenShift** 認証ストラテジーは、Kiali が Red Hat OpenShift Service Mesh (OSSM) とともにデプロイされている場合にサポートされる唯一の認証設定です。**openshift** ストラテジーは、OpenShift Container Platform の個人のロールベースのアクセス制御 (RBAC) ロールに基づいてアクセスを制御します。

7.7.6. 分散トレースのサポートされる設定

- サイドカーとしての Jaeger エージェントは、Jaeger でサポートされる唯一の設定です。デモンセツトとしての Jaeger はマルチテナントインストールまたは OpenShift Dedicated ではサポートされません。

7.7.7. サポート対象の WebAssembly モジュール

- 3scale WebAssembly は、提供されている唯一の WebAssembly モジュールです。カスタム WebAssembly モジュールを作成できます。

7.7.8. Operator の概要

Red Hat OpenShift Service Mesh には、以下の 4 つの Operator が必要です。

- **OpenShift Elasticsearch:**(オプション) 分散トレースプラットフォームでのトレースおよびロギング用にデータベースストレージを提供します。これはオープンソースの [Elasticsearch](#) プロジェクトに基づいています。
- **Red Hat OpenShift 分散トレースプラットフォーム:** 複雑な分散システムでのトランザクションを監視し、トラブルシューティングするための分散トレース機能を提供します。これはオープンソースの [Jaeger](#) プロジェクトに基づいています。
- **Red Hat が提供する Kiali Operator:** サービスメッシュの可観測性を提供します。これにより、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、トレースの分析を実行できます。これはオープンソースの [Kiali](#) プロジェクトに基づいています。
- **Red Hat OpenShift Service Mesh:** 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、観察できます。Service Mesh Operator は、Service Mesh 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。これはオープンソースの [Istio](#) プロジェクトに基づいています。

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。

7.8. ルーティングの最適化

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

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

OpenShift Container Platform Ingress コントローラー (ルーター) は、ルートとイングレスを使用して設定されたアプリケーションとサービスのイングレストラフィックのイングレスポイントです。

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

- HTTP keep-alive/close モード
- ルートタイプ
- TLS セッション再開のクライアントサポート
- ターゲットルートごとの同時接続数

- ターゲットルート数
- バックエンドサーバーのページサイズ
- 基礎となるインフラストラクチャー (ネットワーク/SDN ソリューション、CPU など)

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

HTTP keep-alive モードのシナリオの場合:

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

HTTP close (keep-alive なし) のシナリオの場合:

暗号化	LoadBalancerService	HostNetwork
なし	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

デフォルトの Ingress Controller 設定は、**spec.tuningOptions.threadCount** フィールドを **4** に設定して、使用されました。Load Balancer Service と Host Network という 2 つの異なるエンドポイント公開戦略がテストされました。TLS セッション再開は暗号化ルートについて使用されています。HTTP keep-alive では、1 台の HAProxy ルーターで、8kB という小さなページサイズで 1Gbit の NIC を飽和させることができます。

最新のプロセッサが搭載されたベアメタルで実行する場合は、上記のパブリッククラウドインスタンスのパフォーマンスの約 2 倍のパフォーマンスになることを予想できます。このオーバーヘッドは、パブリッククラウドにある仮想化レイヤーにより発生し、プライベートクラウドベースの仮想化にも多くの場合、該当します。以下の表は、ルーターの背後で使用するアプリケーション数についてのガイドです。

アプリケーション数	アプリケーションタイプ
5-10	静的なファイル/Web サーバーまたはキャッシュプロキシ
100-1000	動的なコンテンツを生成するアプリケーション

通常、HAProxy は、使用しているテクノロジーに応じて、最大 1000 個のアプリケーションのルートをサポートできます。Ingress コントローラーのパフォーマンスは、言語や静的コンテンツと動的コンテンツの違いを含め、その背後にあるアプリケーションの機能およびパフォーマンスによって制限される可能性があります。

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

7.9. インストール後の RHOSP ネットワーク設定

インストール後に、OpenShift Container Platform の一部を Red Hat OpenStack Platform (RHOSP) クラスターに設定することができます。

7.9.1. Floating IP アドレスを使用したアプリケーションアクセスの設定

OpenShift Container Platform をインストールした後に、アプリケーションネットワークトラフィックを許可するように Red Hat OpenStack Platform (RHOSP) を設定します。



注記

インストール中に、**install-config.yaml** ファイルの **platform.openstack.apiFloatingIP** および **platform.openstack.ingressFloatingIP** に値を指定した場合、または **inventory.yaml** Playbook の **os_api_fip** および **os_ingress_fip** に値を指定した場合は、この手順を実行する必要はありません。Floating IP アドレスはすでに設定されています。

前提条件

- OpenShift Container Platform クラスターがインストールされている必要があります。
- OpenShift Container Platform の RHOSP へのインストールに関するドキュメントで説明されているように、Floating IP アドレスが有効にされます。

手順

OpenShift Container Platform クラスターをインストールした後に、Floating IP アドレスを Ingress ポートに割り当てます。

1. ポートを表示します。

```
$ openstack port show <cluster_name>-<cluster_ID>-ingress-port
```

2. ポートを IP アドレスに接続します。

```
$ openstack floating ip set --port <ingress_port_ID> <apps_FIP>
```

3. ***apps.** のワイルドカード **A** レコードを DNS ファイルに追加します。

```
*.apps.<cluster_name>.<base_domain> IN A <apps_FIP>
```

注記

DNS サーバーを制御せず、非実稼働環境でアプリケーションアクセスを有効にする必要がある場合は、これらのホスト名を **/etc/hosts** に追加できます。

```
<apps_FIP> console-openshift-console.apps.<cluster name>.<base domain>
<apps_FIP> integrated-oauth-server-openshift-authentication.apps.<cluster name>.<base domain>
<apps_FIP> oauth-openshift.apps.<cluster name>.<base domain>
<apps_FIP> prometheus-k8s-openshift-monitoring.apps.<cluster name>.<base domain>
<apps_FIP> grafana-openshift-monitoring.apps.<cluster name>.<base domain>
<apps_FIP> <app name>.apps.<cluster name>.<base domain>
```

7.9.2. Kuryr ポートプール

Kuryr ポートプールでは、Pod 作成のスタンバイ状態の多数のポートを維持します。

ポートをスタンバイ状態に維持すると、Pod の作成時間が必要最小限に抑えることができます。ポートプールを使用しない場合には、Kuryr は Pod が作成または削除されるたびにポートの作成または削除を明示的に要求する必要があります。

Kuryr が使用する Neutron ポートは、namespace に関連付けられるサブネットに作成されます。これらの Pod ポートは、OpenShift Container Platform クラスターノードのプライマリーポートにサブポートとして追加されます。

Kuryr は namespace をそれぞれ、別のサブネットに保存するため、namespace-worker ペアごとに別個のポートプールが維持されます。

クラスターをインストールする前に、**cluster-network-03-config.yml** マニフェストファイルに以下のパラメーターを設定して、ポートプールの動作を設定できます。

- **enablePortPoolsPrepopulation** パラメーターは、プールの事前入力を制御します。これにより、Pod 専用ネットワークを使用するように設定された最初の Pod が namespace に作成されたときに、Kuryr が Neutron ポートをプールに追加します。デフォルト値は **false** です。
- **poolMinPorts** パラメーターは、プールに保持する空きポートの最小数です。デフォルト値は **1** です。
- **poolMaxPorts** パラメーターは、プールに保持する空きポートの最大数です。値が **0** の場合は、上限が無効になります。これはデフォルト設定です。
OpenStack ポートのクォータが低い場合や、Pod ネットワークで IP アドレスの数が限定されている場合には、このオプションを設定して、不要なポートが削除されるようにします。
- **poolBatchPorts** パラメーターは、一度に作成可能な Neutron ポートの最大数を定義します。デフォルト値は **3** です。

7.9.3. RHOSP でのアクティブなデプロイメントでの Kuryr ポートプール設定の調整

カスタムリソース (CR) を使用して、Kuryr が Red Hat OpenStack Platform (RHOSP) Neutron ポートをどのように管理するかを設定し、デプロイされたクラスターでの Pod 作成の速度と効率性を制御することができます。

手順

1. コマンドラインから、編集する Cluster Network Operator (CNO) CR を開きます。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 要件に合わせて設定を編集します。以下のファイルをサンプルとして紹介しています。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  serviceNetwork:
    - 172.30.0.0/16
  defaultNetwork:
    type: Kuryr
    kuryrConfig:
      enablePortPoolsPrepopulation: false ❶
      poolMinPorts: 1 ❷
      poolBatchPorts: 3 ❸
      poolMaxPorts: 5 ❹
```

- ❶ **enablePortPoolsPrepopulation** を **true** に設定して、Pod 専用ネットワークを使用するように設定された最初の Pod が namespace で作成されると、Kuryr が Neutron ポートを作成するようにします。この設定により、Neutron ポートのクォータが引き上げられますが、Pod の起動に必要となる時間を短縮できます。デフォルト値は **false** です。
- ❷ Kuryr は、対象のプール内にある空きポートの数が **poolMinPorts** の値よりも少ない場合には、プールに新規ポートを作成します。デフォルト値は **1** です。
- ❸ **poolBatchPorts** は、空きポートの数が **poolMinPorts** の値よりも少ない場合に作成される新規ポートの数を制御します。デフォルト値は **3** です。
- ❹ プール内の空きポートの数が **poolMaxPorts** の値よりも多い場合に、Kuryr はその値と同じ数になるまでポートを削除します。この値を **0** に設定すると、この上限は無効になり、プールが縮小できないようにします。デフォルト値は **0** です。

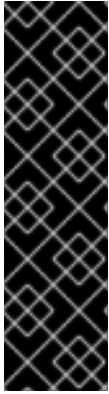
3. 変更を保存し、テキストエディターを終了して、変更をコミットします。



重要

実行中のクラスターでこれらのオプションを変更すると、kuryr-controller および kuryr-cni Pod が再起動を強制的に実行します。その結果、新規 Pod およびサービスの作成が遅延します。

7.9.4. OVS ハードウェアオフロードの有効化



重要

OVS ハードウェアオフロードは、テクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

Red Hat OpenStack Platform (RHOSP) で実行されるクラスターの場合、[Open vSwitch\(OVS\)](#) ハードウェアオフロードを有効にすることができます。

OVS は、大規模なマルチサーバーネットワークの仮想化を可能にするマルチレイヤー仮想スイッチです。

前提条件

- Single-root Input/Output Virtualization (SR-IOV) 用に設定された RHOSP にクラスターをインストールしている。
- SR-IOV Network Operator がクラスターにインストールされている。
- クラスターに 2 つの **hw-offload** タイプの Virtual Function (VF) インターフェイスを作成している。

手順

1. コマンドラインで以下のコマンドを入力し、受付 Webhook を無効にします。

```
$ oc patch sriovoperatorconfig default --type=merge -n openshift-sriov-network-operator --patch '{ "spec": { "enableOperatorWebhook": false } }'
```

2. クラスターにある 2 つの **hw-offload** タイプの VF インターフェイスの **SriovNetworkNodePolicy** ポリシーを作成します。

2 番目の Virtual Function インターフェイス

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy 1
metadata:
  name: "hwoffload9"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: 2
    - ens6
  nodeSelector:
```

```
feature.node.kubernetes.io/network-sriov.capable: 'true'
numVfs: 1
priority: 99
resourceName: "hwoffload9"
```

- ❶ **SriovNetworkNodePolicy** の値をここに挿入します。
- ❷ どちらのインターフェイスにも Physical Function (PF) 名が含まれている必要があります。

2 番目の Virtual Function インターフェイス

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy ❶
metadata:
  name: "hwoffload10"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: ❷
    - ens5
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload10"
```

- ❶ **SriovNetworkNodePolicy** の値をここに挿入します。
- ❷ どちらのインターフェイスにも Physical Function (PF) 名が含まれている必要があります。

3. 2 つのインターフェイス用に **NetworkAttachmentDefinition** リソースを作成します。

1 番目のインターフェイス用 NetworkAttachmentDefinition リソース

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload9
  name: hwoffload9
  namespace: default
spec:
  config: '{"cniVersion":"0.3.1", "name":"hwoffload9", "type":"host-device", "device":"ens6"}'
```

2 番目のインターフェイス用 NetworkAttachmentDefinition リソース

```
apiVersion: k8s.cni.cncf.io/v1
```



```

kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
  name: hwoffload10
  namespace: default
spec:
  config: '{ "cniVersion":"0.3.1", "name":"hwoffload10","type":"host-device","device":"ens5"
}'

```

4. Pod で作成したインターフェイスを使用します。以下に例を示します。

2つの OVS オフロードインターフェイスを使用する Pod

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-testpmd
  namespace: default
  annotations:
    irq-load-balancing.crio.io: disable
    cpu-quota.crio.io: disable
    k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "hwoffload9",
      "namespace": "default"
    },
    {
      "name": "hwoffload10",
      "namespace": "default"
    }
  ]'
spec:
  restartPolicy: Never
  containers:
    - name: dpdk-testpmd
      image: quay.io/krister/centos8_nfv-container-dpdk-testpmd:latest

```

第8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。

8.1. 動的プロビジョニング

8.1.1. 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。

8.1.2. 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	kubernetes.io/cinder	
RHOSP Manila Container Storage Interface (CSI)	manila.csi.openstack.org	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)	kubernetes.io/aws-ebs	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 Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id> のタグを付けます。ここで、 <cluster_name> および <cluster_id> 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	kubernetes.io/azure-disk	
Azure File	kubernetes.io/azure-file	persistent-volume-binder 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)	kubernetes.io/gce-pd	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	kubernetes.io/vsphere-volume	

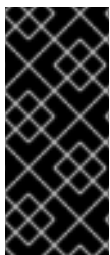


重要

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。

8.2. 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。

8.2.1. 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: <storage-class-name> 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。

8.2.2. 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。ただし、クラスターには複数のストレージクラスを設定できますが、それらのうちの1つのみをデフォルトのストレージクラスにすることができます。



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
...
```

8.2.3. RHOSP Cinder オブジェクトの定義

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> ❶
provisioner: kubernetes.io/cinder
parameters:
  type: fast ❷
  availability: nova ❸
  fsType: ext4 ❹
```

- 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 Cinder で作成されるボリュームタイプ。デフォルトは空です。
- ❸ 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。
- ❹ 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。

8.2.4. 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: <storage-class-name> ❶
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 ❷
  iopsPerGB: "10" ❸
```

```
encrypted: "true" ④
kmsKeyId: keyvalue ⑤
fsType: ext4 ⑥
```

- ① (必須) ストレージクラスの名前。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。
- ② (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 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。
- ③ (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。
- ④ (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。
- ⑤ (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。
- ⑥ (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。

8.2.5. Azure Disk オブジェクト定義

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> ①
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer ②
allowVolumeExpansion: true
parameters:
  kind: Managed ③
  storageaccounttype: Premium_LRS ④
reclaimPolicy: Delete
```

- ① 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。
- ② **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リング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。
- ③ 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。

4

Azure ストレージアカウントの SKU 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 **Standard_LRS** ディスクのみを、マネージド VM はマネージドディスクのみを、アンマネージド VM はアンマネージドディスクのみを割り当てることができます。

- 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。

8.2.6. 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-role>  
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> 1  
provisioner: kubernetes.io/azure-file  
parameters:  
  location: eastus 2  
  skuName: Standard_LRS 3  
  storageAccount: <storage-account> 4  
reclaimPolicy: Delete  
volumeBindingMode: Immediate
```

1 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。

2 **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ます。

3 **Standard_LRS** などの Azure ストレージアカウントの SKU 層。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。

4 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。

8.2.6.1. 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 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

- 1** 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。
- 2** 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。
- 3** シンボリックリンクを有効にします。

8.2.7. 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: <storage-class-name> 1
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 2
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

- 1** 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。
- 2** **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。

8.2.8. VMWare vSphere オブジェクトの定義

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> ❶
provisioner: kubernetes.io/vsphere-volume ❷
parameters:
  diskformat: thin ❸
```

- 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 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。
- ❸ **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。

8.2.9. Red Hat Virtualization (RHV) オブジェクト定義

OpenShift Container Platform は、動的にプロビジョニングされる永続ボリュームを作成するために使用される **ovirt-csi-sc** という名前のタイプが **StorageClass** のデフォルトオブジェクトを作成します。

異なる設定の追加ストレージクラスを作成するには、以下のサンプル YAML で記述される **StorageClass** オブジェクトを使用してファイルを作成し、保存します。

ovirt-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage_class_name> ❶
  annotations:
    storageclass.kubernetes.io/is-default-class: "<boolean>" ❷
provisioner: csi.ovirt.org
allowVolumeExpansion: <boolean> ❸
reclaimPolicy: Delete ❹
volumeBindingMode: Immediate ❺
parameters:
  storageDomainName: <rhv-storage-domain-name> ❻
  thinProvisioning: "<boolean>" ❼
  csi.storage.k8s.io/fstype: <file_system_type> ❽
```

- ❶ ストレージクラス名
- ❷ ストレージクラスがクラスターのデフォルトストレージクラスの場合に **false** に設定されます。**true** に設定される場合、既存のデフォルトストレージクラスを編集し、**false** に設定する必要があります。
- ❸ **true** は動的ボリューム拡張を有効にし、**false** はこれを防ぎます。**true** が推奨されます。
- ❹ このストレージクラスの動的にプロビジョニングされる永続ボリュームは、この回収ポリシーで作成されます。このデフォルトポリシーは **Delete** です。

- 5 **PersistentVolumeClaims** をプロビジョニングし、バインドする方法を示します。設定されていない場合は、**VolumeBindingImmediate** が使用されます。このフィールドは、**VolumeScheduling**
- 6 使用する RHV ストレージドメイン名。
- 7 **true** の場合、ディスクはシンプロビジョニングされます。**false** の場合、ディスクは事前割り当てられます。シンプロビジョニングが推奨されています。
- 8 オプション: 作成するファイルシステムタイプ。使用できる値は **ext4**(デフォルト) または **xfs** です。

8.3. デフォルトストレージクラスの変更

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。たとえば、**gp2** と **standard** の2つのストレージクラスがあり、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ります場合などです。

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. **storageclass.kubernetes.io/is-default-class** アノテーションを **true** に設定して、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

8.4. 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。

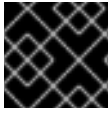
8.5. 利用可能な永続ストレージオプション

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8.1 利用可能なストレージオプション

ストレージタイプ	説明	例
ブロック	- ブロックデバイスとしてオペレーティングシステムに公開されます。 - ストレージを完全に制御し、ファイルシステムを通過してファイルの低いレベルで操作する必要があるアプリケーションに適しています。 - ストレージエリアネットワーク (SAN) とも呼ばれます。 - 共有できません。一度に1つのクライアントだけがこのタイプのエンドポイントをマウントできるという意味です。	AWS EBS および VMware vSphere は、OpenShift Container Platform で永続ボリューム (PV) の動的なプロビジョニングをサポートします。
ファイル	- マウントされるファイルシステムのエクスポートとして、OS に公開されます。 - ネットワークアタッチストレージ (NAS) とも呼ばれます。 - 同時実行、レイテンシー、ファイルロックのメカニズムその他の各種機能は、プロトコルおよび実装、ベンダー、スケールによって大きく異なります。	RHEL NFS、NetApp NFS ^[1] 、および Vendor NFS
オブジェクト	- REST API エンドポイント経由でアクセスできます。 - OpenShift イメージレジストリーで使用するように設定できます。 - アプリケーションは、ドライバーをアプリケーションやコンテナに組み込む必要があります。	AWS S3

1. NetApp NFS は Trident を使用する場合に動的 PV のプロビジョニングをサポートします。

**重要**

現時点で、CNS は OpenShift Container Platform 4.10 ではサポートされていません。

8.6. 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8.2 設定可能な推奨ストレージ技術

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

¹ **ReadOnlyMany**

² **ReadWriteMany**

³ Prometheus はメトリクスに使用される基礎となるテクノロジーです。

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

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

⁶ ロギングの場合、共有ストレージを使用することはアンチパターンとなります。elasticsearch ごとに 1 つのボリュームが必要です。

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

**注記**

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

8.6.1. 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される PV をサポートするために RHEL NFS を使用することは推奨されていません。

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

8.6.1.1. レジストリー

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

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

8.6.1.2. スケーリングされたレジストリー

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

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

8.6.1.3. 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。

8.6.1.4. 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:

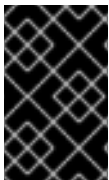
- 推奨されるストレージ技術はブロックストレージです。
- オブジェクトストレージは設定できません。

8.6.1.5. 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ルアップ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。

8.6.2. 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



重要

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

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder は ROX アクセスモードのユースケースで適切に機能する傾向があります。
- データベース: データベース (RDBMS、NoSQL DB など) は、専用のブロックストレージで最適に機能することが予想されます。
- etcd データベースには、大規模なクラスターを有効にするのに十分なストレージと十分なパフォーマンス容量が必要です。十分なストレージと高性能環境を確立するための監視およびベンチマークツールに関する情報は、**推奨される etcd プラクティス**に記載されています。

関連情報

- [etcd についての推奨されるプラクティス](#)

8.7. RED HAT OPENSIFT DATA FOUNDATION のデプロイ

Red Hat OpenShift Data Foundation 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 Data Foundation 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 Data Foundation に関する情報	Red Hat OpenShift Data Foundation の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	OpenShift Data Foundation 4.9 リリースノート
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	OpenShift Data Foundation 4.9 デプロイメントの計画
外部の Red Hat Ceph Storage クラスターを使用するように OpenShift Data Foundation をデプロイする手順	外部モードでの OpenShift Data Foundation 4.9 のデプロイ
ベアメタルインフラストラクチャーでローカルストレージを使用した OpenShift Container Storage のデプロイ手順	ベアメタルインフラストラクチャーを使用した OpenShift Data Foundation 4.9 のデプロイ
Red Hat OpenShift Container Platform VMware vSphere クラスターへの OpenShift Data Foundation のデプロイ手順	VMware vSphere 上での OpenShift Data Foundation 4.9 のデプロイ
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 Data Foundation のデプロイ手順	Amazon Web サービスを使用した OpenShift Data Foundation 4.9 のデプロイ
既存の Red Hat OpenShift Container Platform Google Cloud クラスターへの OpenShift Data Foundation のデプロイおよび管理手順	Google Cloud を使用した OpenShift Data Foundation 4.9 のデプロイおよび管理
既存の Red Hat OpenShift Container Platform Azure クラスターへの OpenShift Data Foundation のデプロイおよび管理手順	Microsoft Azure を使用した OpenShift Data Foundation 4.9 のデプロイおよび管理
IBM Power インフラストラクチャーでローカルストレージを使用する OpenShift Data Foundation のデプロイ手順	IBM Power での OpenShift Data Foundation のデプロイ
IBM Z インフラストラクチャーでローカルストレージを使用する OpenShift Data Foundation のデプロイ手順	IBM Z インフラストラクチャーへの OpenShift Data Foundation のデプロイ
スナップショットおよびクローンを含む、Red Hat OpenShift Data Foundation のコアサービスおよびホスト型アプリケーションへのストレージの割り当て	リソースの管理および割り当て

Red Hat OpenShift Data Foundation に関する情報	Red Hat OpenShift Data Foundation のドキュメントの参照先
Multicloud Object Gateway (NooBaa) を使用したハイブリッドクラウドまたはマルチクラウド環境でのストレージリソースの管理	ハイブリッドおよびマルチクラウドリソースの管理
Red Hat OpenShift Data Foundation のストレージデバイスの安全な置き換え	デバイスの置き換え
Red Hat OpenShift Data Foundation クラスター内のノードの安全な置き換え	ノードの置き換え
Red Hat OpenShift Data Foundation でのスケーリング操作	ストレージのスケーリング
Red Hat OpenShift Data Foundation 4.9 クラスターのモニタリング	Red Hat OpenShift Data Foundation 4.9 のモニタリング
操作中に発生する問題の解決	OpenShift Data Foundation 4.9 のトラブルシューティング
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	移行

第9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。

9.1. 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。

9.1.1. 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。

9.1.2. 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	htpasswd アイデンティティプロバイダーを htpasswd 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	keystone 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	ldap 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	basic-authentication 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用して 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	request-header アイデンティティプロバイダーを、 X-Remote-User 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
GitHub または GitHub Enterprise	github 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。
GitLab	gitlab アイデンティティプロバイダーを、 GitLab.com 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。
Google	google アイデンティティプロバイダーを、 Google の OpenID Connect 統合 を使用して設定します。
OpenID Connect	oidc アイデンティティプロバイダーを、 Authorization Code Flow 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。

9.1.3. 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
name	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
mappingMethod	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。 claim 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。 lookup 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。 generate 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: myuser2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。 add 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。



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。

9.1.4. 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。

9.2. 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。

9.2.1. 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 create 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	ロールを使用したユーザー/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。 クラスターロール はクラスター全体で存在し、 クラスターロールのバインディング 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。 ローカルロール 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 両方 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。

9.2.1.1. 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。



重要

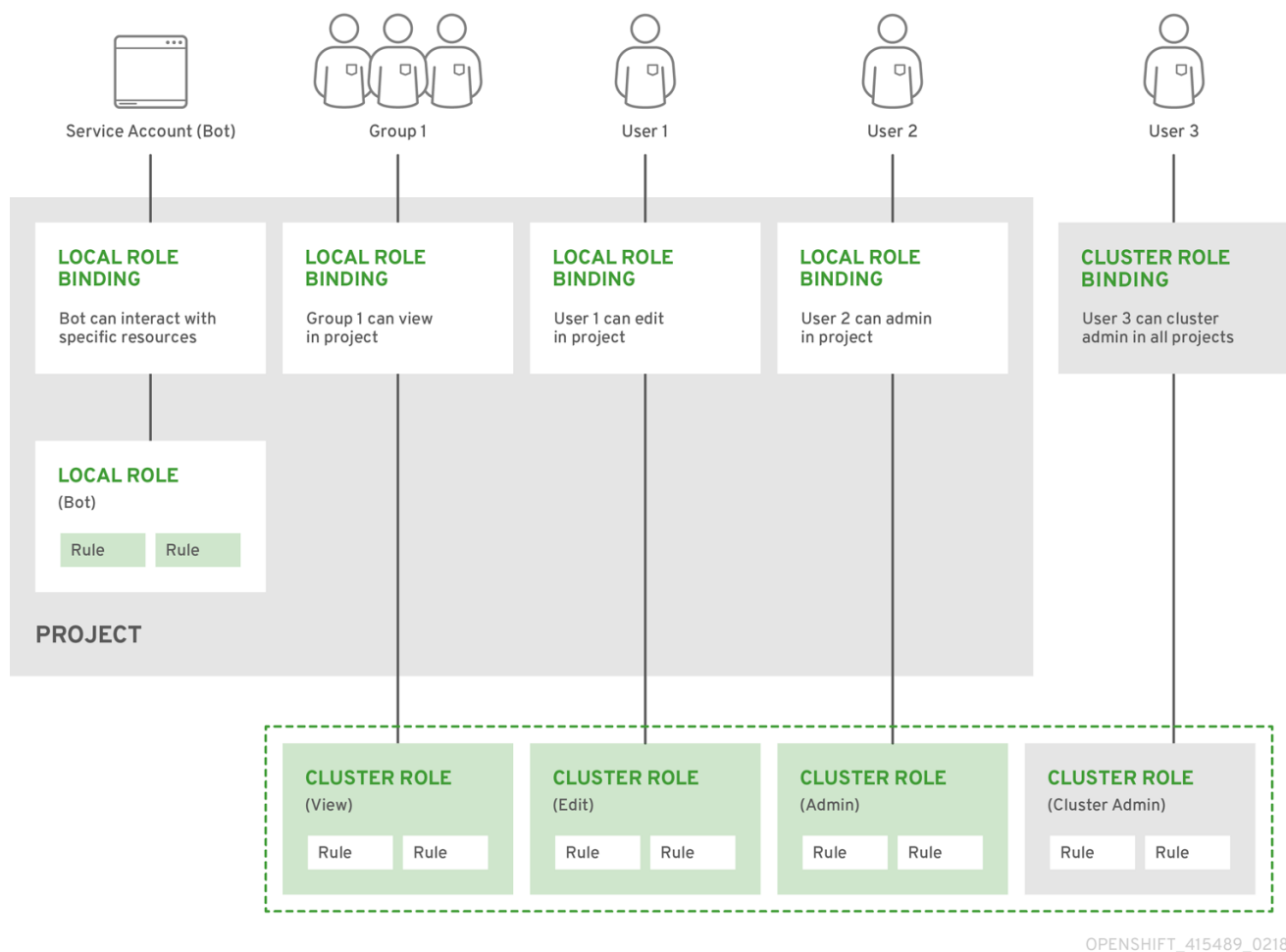
デフォルトのクラスターロールを手動で変更することは推奨されません。このようなシステムロールへの変更は、クラスターが正常に機能しなくなることがあります。

デフォルトのクラスターロール	説明
admin	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、 admin 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。
basic-user	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。
cluster-admin	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。

デフォルトのクラスターロール	説明
cluster-status	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。
cluster-reader	ほとんどのオブジェクトを取得または表示できるが、変更できないユーザー。
edit	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。
self-provisioner	独自のプロジェクトを作成できるユーザーです。
view	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。



警告

get pods/exec、**get pods/***、および **get *** ルールは、ロールに適用されると実行権限を付与します。最小権限の原則を適用し、ユーザーおよびエージェントに必要な最小限の RBAC 権限のみを割り当てます。詳細は、[RBAC ルールによる実行権限の許可](#) を参照してください。

9.2.1.2. 認可の評価

OpenShift Container Platform は以下を使用して認可を評価します。

アイデンティティ

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。

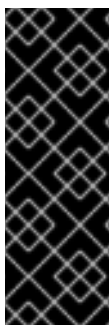
OpenShift Container Platform は以下の手順を使用して認可を評価します。

1. アイデンティティ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。

9.2.1.2.1. クラスターロールの集計

デフォルトの admin、edit、view、cluster-reader クラスターロールでは、[クラスターロールの集約](#) がサポートされており、各ロールは新規ルール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。

9.2.2. 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
Objects	Pod、サービス、レプリケーションコントローラーなど。
Policies	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。
Constraints	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。
Service accounts	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。

9.2.3. 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。

9.2.4. 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 がインストールされている。
- 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]
deletecollection patch update]			
Pods/exec	[]	[]	[get list watch create delete deletecollection patch update]
deletecollection patch update]			
Pods/portforward	[]	[]	[get list watch create delete deletecollection patch update]
delete deletecollection patch update]			
Pods/proxy	[]	[]	[get list watch create delete deletecollection patch update]
deletecollection patch update]			
services/proxy	[]	[]	[get list watch create delete deletecollection patch update]
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

...

```

9.2.5. 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 がインストールされている。
- 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

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	

Group system:serviceaccounts:joe-project

9.2.6. 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、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
```

ヒント

または、以下の YAML を適用してユーザーにロールを追加できます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: admin-0
  namespace: joe
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: alice
```

- 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。

9.2.7. 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
```

9.2.8. 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
```

9.2.9. 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9.1 ローカルのロールバインディング操作

コマンド	説明
\$ oc adm policy who-can <verb> <resource>	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。
\$ oc adm policy add-role-to-user <role> <username>	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。
\$ oc adm policy remove-role-from-user <role> <username>	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。
\$ oc adm policy remove-user <username>	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。
\$ oc adm policy add-role-to-group <role> <groupname>	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。
\$ oc adm policy remove-role-from-group <role> <groupname>	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。
\$ oc adm policy remove-group <groupname>	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。

9.2.10. 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9.2 クラスターのロールバインディング操作

コマンド	説明
\$ oc adm policy add-cluster-role-to-user <role> <username>	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。
\$ oc adm policy remove-cluster-role-from-user <role> <username>	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。
\$ oc adm policy add-cluster-role-to-group <role> <groupname>	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。
\$ oc adm policy remove-cluster-role-from-group <role> <groupname>	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。

9.2.11. 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>
```

9.3. 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-abc.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

9.3.1. kubeadmin ユーザーの削除

アイデンティティ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
```

9.4. イメージ設定

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。

9.4.1. 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。



注記

DisableScheduledImport、**MaxImagesBulkImportedPerRepository**、**MaxScheduledImportsPerMinute**、**ScheduledImageImportMinimumIntervalSeconds**、**InternalRegistryHostname** などのパラメーターは設定できません。

パラメーター	説明
allowedRegistriesForImport	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 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。 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。 domainName: 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 (80 または 443) 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。 insecure: insecure はレジストリーがセキュアか、非セキュアであるかを示します。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。
additionalTrustedCA	image stream import、pod image pull、openshift-image-registry pullthrough、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。 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 エンコード証明書を値として使用します。

パラメーター	説明
externalRegistryHostnames	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 publicDockerImageRepository フィールドで使用されます。値は hostname[:port] 形式の値である必要があります。
registrySources	コンテナー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。 insecureRegistries: 有効な TLS 証明書を持たないか、HTTP 接続のみをサポートするレジストリーです。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: *.example.com レジストリー内で個別のリポジトリを指定できます。例: reg1.io/myrepo/myapp:latest blockedRegistries: イメージのプルおよびプッシュアクションが拒否されるレジストリーです。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: *.example.com レジストリー内で個別のリポジトリを指定できます。例: reg1.io/myrepo/myapp:latest 他のすべてのレジストリーは許可されます。 allowedRegistries: イメージのプルおよびプッシュアクションが許可されるレジストリーです。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: *.example.com レジストリー内で個別のリポジトリを指定できます。例: reg1.io/myrepo/myapp:latest 他のすべてのレジストリーはブロックされます。 containerRuntimeSearchRegistries: イメージの短縮名を使用したイメージのプルおよびプッシュアクションが許可されるレジストリーです。他のすべてのレジストリーはブロックされます。 blockedRegistries または allowedRegistries 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。



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 OpenShift 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
internalRegistryHostname	internalRegistryHostname 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 OpenShift イメージレジストリーのホスト名を設定します。値は hostname[:port] 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、 OPENSIFT_DEFAULT_REGISTRY 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。
externalRegistryHostnames	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 publicDockerImageRepository フィールドで使用されます。値は hostname[:port] 形式の値である必要があります。

9.4.2. イメージレジストリーの設定

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。レジストリーへの変更が **image.config.openshift.io/cluster** CR に適用されると、Machine Config Operator (MCO) は以下の一連のアクションを実行します。

1. ノードを封鎖します
2. CRI-O を再起動して変更を適用します
3. ノードを解放します



注記

MCO は、変更を検出してもノードを再起動しません。

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 ①
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
      - reg1.io/myrepo/myapp:latest
    insecureRegistries:
      - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- ❶ **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。
- ❷ **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。
- ❸ **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。
- ❹ **registrySources:** ビルドおよび Pod のイメージにアクセスする際に、コンテナランタイムが個々のレジストリーを許可するかブロックするかを決定する設定が含まれます。**allowedRegistries** パラメーターまたは **blockedRegistries** パラメーター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。安全でないレジストリーまたはイメージの短い名前を使用するレジストリーを許可するレジストリーへのアクセスを許可するかどうかを定義することもできます。この例では、使用が許可されるレジストリーを定義する **allowedRegistries** パラメーターを使用します。安全でないレジストリー **insecure.com** も許可されます。**registrySources** パラメーターには、内部クラスターレジストリーの設定は含まれません。



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 OpenShift 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 **allowedRegistries** 一覧に追加する必要があります。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。**registry.redhat.io** および **quay.io** レジストリーを **blockedRegistries** 一覧に追加しないでください。

allowedRegistries、**blockedRegistries**、または **insecureRegistries** パラメーターを使用する場合、レジストリー内に個別のリポジトリを指定できます。
例: **reg1.io/myrepo/myapp:latest**

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。

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
ip-10-0-137-182.us-east-2.compute.internal	Ready,SchedulingDisabled	worker	65m	v1.25.4+77bec7a
ip-10-0-139-120.us-east-2.compute.internal	Ready,SchedulingDisabled	control-plane	74m	v1.25.4+77bec7a
ip-10-0-176-102.us-east-2.compute.internal	Ready	control-plane	75m	v1.25.4+77bec7a
ip-10-0-188-96.us-east-2.compute.internal	Ready	worker	65m	v1.25.4+77bec7a
ip-10-0-200-59.us-east-2.compute.internal	Ready	worker	63m	v1.25.4+77bec7a
ip-10-0-223-123.us-east-2.compute.internal	Ready	control-plane	73m	v1.25.4+77bec7a

許可、ブロック、および安全でないレジストリーパラメーターの詳細は、[イメージレジストリーの設定](#)を参照してください。

9.4.2.1. イメージレジストリ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 config map 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。

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 を指定できます。

設定マップキーは、この CA を信頼するポートがあるレジストリーのホスト名であり、値は各追加レジストリー CA が信頼する証明書のコンテンツです。

イメージレジストリー CA の config map 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
```

9.4.2.2. 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りポジトリを特定し、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。
- 切断された環境のクラスターは、quay.io などの重要な場所からイメージをプルし、会社のファイアウォールの背後にあるレジストリーに要求されたイメージを提供することができます。

- 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 に必要な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背後に配置することで、切断された環境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ー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
    - example.com/example/ubi-minimal ❷
    source: registry.access.redhat.com/ubi8/ubi-minimal ❸
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 ❹
  - mirrors:
    - mirror.example.com
    source: registry.redhat.io ❺
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage ❻
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example ❼
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com ❽
```

- 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 各ターゲットリポジトリの複数のミラーリポジトリを示します。1つのミラーがダウンした場合、ターゲットリポジトリは別のミラーを使用できます。
- ❸ 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。
- ❹ レジストリー内の namespace を、その namespace の任意のイメージを使用するように設定できます。レジストリードメインをソースとして使用する場合は、**ImageContentSourcePolicy** リソースはレジストリーからすべてのリポジトリに適用されます。

- 5 レジストリー名を設定すると、ソースレジストリーからミラーレジストリーまでのすべてのリポジトリに **ImageContentSourcePolicy** リソースが適用されます。
- 6 イメージ **mirror.example.net/image@sha256:...** をプルします。
- 7 ミラー **mirror.example.net/myimage@sha256:...** からソースレジストリー namespace のイメージ **myimage** をプルします。
- 8 ミラーレジストリー **mirror.example.net/registry-example-com/example/myimage@sha256:...** からイメージ **registry.example.com/example/myimage** をプルします。**ImageContentSourcePolicy** リソースは、ソースレジストリーからミラーレジストリー **mirror.example.net/registry-example-com** までのすべてのリポジトリに適用されます。

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	v1.24.0
ip-10-0-138-148.ec2.internal	Ready	master	11m	v1.24.0
ip-10-0-139-122.ec2.internal	Ready	master	11m	v1.24.0
ip-10-0-147-35.ec2.internal	Ready	worker	7m	v1.24.0
ip-10-0-153-12.ec2.internal	Ready	worker	7m	v1.24.0
ip-10-0-154-10.ec2.internal	Ready	master	11m	v1.24.0

Imagecontentsourcepolicy リソースはノードを再起動しません。

- 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. ルートディレクトリーを **/host** に変更します。

```
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"]
short-name-mode = ""
```

```
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi8/ubi-minimal"
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "example.io/example/ubi-minimal"
```

```
[[registry.mirror]]
  location = "example.com/example/ubi-minimal"
```

```
[[registry]]
  prefix = ""
  location = "registry.example.com"
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "mirror.example.net/registry-example-com"
```

```
[[registry]]
  prefix = ""
  location = "registry.example.com/example"
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "mirror.example.net"
```

```
[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "mirror.example.net/image"
```

```
[[registry]]
  prefix = ""
  location = "registry.redhat.io"
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "mirror.example.com"
```

```
[[registry]]
  prefix = ""
```

```
location = "registry.redhat.io/openshift4"
mirror-by-digest-only = true

[[registry.mirror]]
location = "mirror.example.com/redhat"
```

- 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@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba6
```

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。

9.5. ミラーリングされた OPERATOR カタログからの OPERATORHUB の入力

非接続クラスターで使用するように Operator カタログをミラーリングする場合は、OperatorHub をミラーリングされたカタログの Operator で設定できます。ミラーリングプロセスから生成されたマニフェストを使用して、必要な **ImageContentSourcePolicy** および **CatalogSource** オブジェクトを作成できます。

9.5.1. 前提条件

- [非接続クラスターで使用する Operator カタログのミラーリング](#)

9.5.2. ImageContentSourcePolicy オブジェクトの作成

Operator カタログコンテンツをミラーレジストリーにミラーリングした後に、必要な **ImageContentSourcePolicy** (ICSP) オブジェクトを作成します。ICSP オブジェクトは、Operator マニフェストおよびミラーリングされたレジストリーに保存されるイメージ参照間に変換するようにノードを設定します。

手順

- 非接続クラスターへのアクセスのあるホストで、以下のコマンドを実行して manifests ディレクトリで **imageContentSourcePolicy.yaml** ファイルを指定して ICSP を作成します。

```
$ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
```

ここで、**<path/to/manifests/dir>** は、ミラーリングされたコンテンツについての manifests ディレクトリーへのパスです。

ミラーリングされたインデックスイメージおよび Operator コンテンツを参照する **CatalogSource** を作成できるようになりました。

9.5.3. クラスターへのカタログソースの追加

カタログソースを OpenShift Container Platform クラスターに追加すると、ユーザーの Operator の検出およびインストールが可能になります。クラスター管理者は、インデックスイメージを参照する **CatalogSource** オブジェクトを作成できます。OperatorHub はカタログソースを使用してユーザーインターフェイスを設定します。

前提条件

- レジストリーにビルドされ、プッシュされるインデックスイメージ。

手順

1. インデックスイメージを参照する **CatalogSource** オブジェクトを作成します。 **oc adm catalog mirror** コマンドを使用してカタログをターゲットレジストリーにミラーリングする場合、manifests ディレクトリーに生成される **catalogSource.yaml** ファイルを開始点としてそのまま使用することができます。
  - a. 仕様を以下のように変更し、これを **catalogSource.yaml** ファイルとして保存します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog ①
  namespace: openshift-marketplace ②
spec:
  sourceType: grpc
  image: <registry>/<namespace>/redhat-operator-index:v4.10 ③
  displayName: My Operator Catalog
  publisher: <publisher_name> ④
  updateStrategy:
    registryPoll: ⑤
    interval: 30m
```

- ① レジストリーにアップロードする前にローカルファイルにコンテンツをミラーリングする場合は、**metadata.name** フィールドからバックスラッシュ (`/`) 文字を削除し、オブジェクトの作成時に invalid resource name エラーを回避します。
- ② カatalogソースを全 namespace のユーザーがグローバルに利用できるようにする場合は、**openshift-marketplace** namespace を指定します。それ以外の場合は、そのカタログの別の namespace を対象とし、その namespace のみが利用できるように指定できます。
- ③ インデックスイメージを指定します。イメージ名の後にタグを指定した場合 (**:v4.10** など)、カタログソース Pod は **Always** のイメージプルポリシーを使用します。これは、Pod が常にコンテナを開始する前にイメージをプルすることを意味します。**@sha256:<id>** などのダイジェストを指定した場合、イメージプルポリシーは **IfNotPresent** になります。これは、イメージがノード上にまだ存在しない場合にのみ、Pod がイメージをプルすることを意味します。

- 4 カタログを公開する名前または組織名を指定します。
- 5 カタログソースは新規バージョンの有無を自動的にチェックし、最新の状態を維持します。

b. このファイルを使用して **CatalogSource** オブジェクトを作成します。

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

2. 以下のリソースが正常に作成されていることを確認します。

a. Pod を確認します。

```
$ oc get pods -n openshift-marketplace
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
my-operator-catalog-6njsx6          1/1   Running 0      28s
marketplace-operator-d9f549946-96sgr 1/1   Running 0      26h
```

b. カタログソースを確認します。

```
$ oc get catalogsource -n openshift-marketplace
```

出力例

```
NAME          DISPLAY          TYPE PUBLISHER AGE
my-operator-catalog My Operator Catalog grpc      5s
```

c. パッケージマニフェストを確認します。

```
$ oc get packagemanifest -n openshift-marketplace
```

出力例

```
NAME          CATALOG          AGE
jaeger-product My Operator Catalog 93s
```

OpenShift Container Platform Web コンソールで、**OperatorHub** ページから Operator をインストールできるようになりました。

関連情報

- [プライベートレジストリーからの Operator のイメージへのアクセス](#)
- [カスタムカタログソースのイメージテンプレート](#)
- [イメージプルポリシー](#)

9.6. OPERATORHUB を使用した OPERATOR のインストールについて

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、(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。

9.6.1. 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、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。

9.6.2. 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 **SingleNamespaces** 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 **SingleNamespaces** 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
  config:
    env: ❻
    - name: ARGS
      value: "-v=10"
    envFrom: ❼
    - secretRef:
        name: license-secret
  volumes: ❽
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: ❾
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: ❿
```

```
- operator: "Exists"
resources: 11
  requests:
    memory: "64Mi"
    cpu: "250m"
  limits:
    memory: "128Mi"
    cpu: "500m"
nodeSelector: 12
  foo: bar
```

- 1 **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。
- 6 **env** パラメーターは、OLM によって作成される Pod のすべてのコンテナに存在する必要がある環境変数の一覧を定義します。
- 7 **envFrom** パラメーターは、コンテナの環境変数に反映するためのソースの一覧を定義します。
- 8 **volumes** パラメーターは、OLM によって作成される Pod に存在する必要があるボリュームの一覧を定義します。
- 9 **volumeMounts** パラメーターは、OLM によって作成される Pod のすべてのコンテナに存在する必要があるボリュームマウントの一覧を定義します。**volumeMount** が存在しない **ボリューム** を参照する場合、OLM は Operator のデプロイに失敗します。
- 10 **tolerations** パラメーターは、OLM によって作成される Pod の容認の一覧を定義します。
- 11 **resources** パラメーターは、OLM によって作成される Pod のすべてのコンテナのリソース制約を定義します。
- 12 **nodeSelector** パラメーターは、OLM によって作成される Pod の **ノードセレクター** を定義します。

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](#)

第10章 アラート通知の設定

OpenShift Container Platform では、アラートは、アラートルールで定義された条件が true の場合に実行されます。アラートは、一連の状況がクラスター内で明確であることを示す通知を提供します。実行するアラートは、OpenShift Container Platform web コンソールでデフォルトで表示できます。インストール後に、OpenShift Container Platform を外部システムにアラート通知を送信するように設定できます。

10.1. 外部システムへの通知の送信

OpenShift Container Platform 4.10 では、実行するアラートをアラート UI で表示できます。アラートは、デフォルトでは通知システムに送信されるように設定されません。以下のレシーバータイプにアラートを送信するように OpenShift Container Platform を設定できます。

- PagerDuty
- Webhook
- Email
- Slack

レシーバーへのアラートのルートを指定することにより、障害が発生する際に適切なチームに通知をタイムリーに送信できます。たとえば、重大なアラートには早急な対応が必要となり、通常は個人または緊急対策チーム (Critical Response Team) に送信先が設定されます。重大ではない警告通知を提供するアラートは、早急な対応を要さないレビュー用にチケットシステムにルート指定される可能性があります。

Watchdog アラートの使用によるアラートが機能することの確認

OpenShift Container Platform モニタリングには、継続的に実行される Watchdog アラートが含まれます。Alertmanager は、Watchdog のアラート通知を設定された通知プロバイダーに繰り返し送信します。通常、プロバイダーは Watchdog アラートの受信を停止する際に管理者に通知するように設定されます。このメカニズムは、Alertmanager と通知プロバイダー間の通信に関連する問題を迅速に特定するのに役立ちます。

10.1.1. アラートレシーバーの設定

アラートレシーバーを設定して、クラスターについての重要な問題について把握できるようにします。

前提条件

- **cluster-admin** クラスター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ます。

手順

1. **Administrator** パースペクティブで、**Administration** → **Cluster Settings** → **Configuration** → **Alertmanager** に移動します。



注記

または、通知ドロワーで同じページに移動することもできます。OpenShift Container Platform Web コンソールの右上にあるベルのアイコンを選択し、**AlertmanagerReceiverNotConfigured** アラートで **Configure** を選択します。

2. ページの **Receivers** セクションで **Create Receiver** を選択します。
3. **Create Receiver** フォームで、**Receiver Name** を追加し、一覧から **Receiver Type** を選択します。
4. レシーバー設定を編集します。
  - PagerDuty receiver の場合:
    - a. 統合のタイプを選択し、PagerDuty 統合キーを追加します。
    - b. PagerDuty インストールの URL を追加します。
    - c. クライアントおよびインシデントの詳細または重大度の指定を編集する場合は、**Show advanced configuration** を選択します。
  - Webhook receiver の場合:
    - a. HTTP POST リクエストを送信するエンドポイントを追加します。
    - b. デフォルトオプションを編集して解決したアラートを receiver に送信する場合は、**Show advanced configuration** を選択します。
  - メール receiver の場合:
    - a. 通知を送信するメールアドレスを追加します。
    - b. SMTP 設定の詳細を追加します。これには、通知の送信先のアドレス、メールの送信に使用する smarthost およびポート番号、SMTP サーバーのホスト名、および認証情報を含む詳細情報が含まれます。
    - c. TLS が必要であるかどうかについて選択します。
    - d. デフォルトオプションを編集して解決済みのアラートが receiver に送信されないようにしたり、メール通知設定の本体を編集する必要がある場合は、**Show advanced configuration** を選択します。
  - Slack receiver の場合:
    - a. Slack Webhook の URL を追加します。
    - b. 通知を送信する Slack チャンネルまたはユーザー名を追加します。
    - c. デフォルトオプションを編集して解決済みのアラートが receiver に送信されないようにしたり、アイコンおよびユーザー設定を編集する必要がある場合は、**Show advanced configuration** を選択します。チャンネル名とユーザー名を検索し、これらをリンクするかどうかについて選択することもできます。
5. デフォルトでは、実行されるすべてのセクターに一致するラベルを持つアラートは receiver に送信されます。実行されるアラートのラベルの値を、それらが receiver に送信される前の状態に完全に一致させる必要がある場合は、以下を実行します。
  - a. ルーティングラベル名と値をフォームの **Routing Labels** セクションに追加します。
  - b. 正規表現を使用する場合は **Regular Expression** を選択します。
  - c. **Add Label** を選択して、さらにルーティングラベルを追加します。
6. **Create** を選択して receiver を作成します。

10.2. 関連情報

- [モニタリングの概要](#)
- [アラートの管理](#)

第11章 接続クラスターの非接続クラスターへの変換

OpenShift Container Platform クラスターを接続クラスターから非接続クラスターに変換する必要のあるシナリオがある場合があります。

制限されたクラスターとも呼ばれる非接続クラスターには、インターネットへのアクティブな接続がありません。そのため、レジストリーおよびインストールメディアのコンテンツをミラーリングする必要があります。インターネットと閉じられたネットワークの両方にアクセスできるホスト上にこのミラーレジストリーを作成したり、ネットワークの境界を越えて移動できるデバイスにイメージをコピーしたりすることができます。

このトピックでは、既存の接続クラスターを非接続クラスターに変換する一般的なプロセスについて説明します。

11.1. ミラーレジストリーについて

OpenShift Container Platform のインストールとその後の製品更新に必要なイメージは、Red Hat Quay、JFrog Artifactory、Sonatype Nexus Repository、Harbor などのコンテナミラーレジストリーにミラーリングできます。大規模なコンテナレジストリーにアクセスできない場合は、OpenShift Container Platform サブスクリプションに含まれる小規模なコンテナレジストリーである **Red Hat OpenShift 導入用のミラーレジストリー** を使用できます。

Red Hat Quay、**Red Hat OpenShift 導入用のミラーレジストリー**、Artifactory、Sonatype Nexus リポジトリ、Harbor など、[Docker v2-2](#) をサポートする任意のコンテナレジストリーを使用できます。選択したレジストリーに関係なく、インターネット上の Red Hat がホストするサイトから分離されたイメージレジストリーにコンテンツをミラーリングする手順は同じです。コンテンツをミラーリングした後に、各クラスターをミラーレジストリーからこのコンテンツを取得するように設定します。



重要

OpenShift イメージレジストリーはターゲットレジストリーとして使用できません。これは、ミラーリングプロセスで必要となるタグを使わないプッシュをサポートしないためです。

Red Hat OpenShift 導入用のミラーレジストリー以外のコンテナレジストリーを選択する場合は、プロビジョニングするクラスター内の全マシンから到達可能である必要があります。レジストリーに到達できない場合、インストール、更新、またはワークロードの再配置などの通常の操作が失敗する可能性があります。そのため、ミラーレジストリーは可用性の高い方法で実行し、ミラーレジストリーは少なくとも OpenShift Container Platform クラスターの実稼働環境の可用性の条件に一致している必要があります。

ミラーレジストリーを OpenShift Container Platform イメージで設定する場合、2つのシナリオを実行することができます。インターネットとミラーレジストリーの両方にアクセスできるホストがあり、クラスターノードにアクセスできない場合は、そのマシンからコンテンツを直接ミラーリングできます。このプロセスは、**connected mirroring** (接続ミラーリング) と呼ばれます。このようなホストがない場合は、イメージをファイルシステムにミラーリングしてから、そのホストまたはリムーバブルメディアを制限された環境に配置する必要があります。このプロセスは、**disconnected mirroring** (非接続ミラーリング) と呼ばれます。

ミラーリングされたレジストリーの場合は、プルされたイメージのソースを表示するには、CRI-O ログで **Trying to access** のログエントリーを確認する必要があります。ノードで **crictl images** コマンドを使用するなど、イメージのプルソースを表示する他の方法では、イメージがミラーリングされた場所からプルされている場合でも、ミラーリングされていないイメージ名を表示します。



注記

Red Hat は、OpenShift Container Platform を使用してサードパーティーのレジストリーをテストしません。

11.2. 前提条件

- **oc** クライアントがインストールされている。
- 実行中のクラスター。
- OpenShift Container Platform クラスターをホストする場所で [Docker v2-2](#) をサポートするコンテナイメージレジストリーであるミラーレジストリーがインストールされている (例: 以下のレジストリーのいずれか)。
  - [Red Hat Quay](#)
  - [JFrog Artifactory](#)
  - [Sonatype Nexus](#) リポジトリ
  - [Harbor](#)

Red Hat Quay のサブスクリプションをお持ちの場合は、Red Hat Quay のデプロイに関するドキュメント [概念実証 \(実稼働以外\) 向けの Red Hat Quay のデプロイ](#) または [Quay Operator の使用による OpenShift への Red Hat Quay のデプロイ](#) を参照してください。

- ミラーリポジトリは、イメージを共有するように設定される必要があります。たとえば、Red Hat Quay リポジトリでは、イメージを共有するために [Organizations](#) が必要です。
- 必要なコンテナイメージを取得するためのインターネットへのアクセス。

11.3. ミラーリングのためのクラスターの準備

クラスターの接続を切断する前に、非接続クラスター内のすべてのノードから到達可能なミラーレジストリーにイメージをミラーリングまたはコピーする必要があります。イメージをミラーリングするには、以下を実行してクラスターを準備する必要があります。

- ミラーレジストリー証明書をホストの信頼される CA の一覧に追加する。
- **cloud.openshift.com** トークンからのイメージプルシークレットが含まれる **.dockerconfigjson** ファイルを作成する。

手順

1. イメージのミラーリングを可能にする認証情報を設定します。
  - a. 単純な PEM または DER ファイル形式で、ミラーレジストリーの CA 証明書を信頼される CA の一覧に追加します。以下に例を示します。

```
$ cp </path/to/cert.crt> /usr/share/pki/ca-trust-source/anchors/
```

ここでは、以下ようになります。 **</path/to/cert.crt>**

ローカルファイルシステムの証明書へのパスを指定します。

- b. CA 信頼を更新します。たとえば、Linux の場合は以下ようになります。

```
$ update-ca-trust
```

- c. グローバルプルシークレットから **.dockerconfigjson** ファイルを展開します。

```
$ oc extract secret/pull-secret -n openshift-config --confirm --to=.
```

出力例

```
.dockerconfigjson
```

- d. **.dockerconfigjson** ファイルを編集し、ミラーレジストリーおよび認証情報を追加し、これを新規ファイルとして保存します。

```
{"auths":{"<local_registry>":{"auth":"<credentials>","email":"you@example.com"}},"<registry>:<port>/<namespace>/{\"auth\":\"<token>\"}}}
```

ここでは、以下ようになります。

<local_registry>

ミラーレジストリーがコンテンツを提供するために使用するレジストリーのドメイン名およびポート (オプション) を指定します。

auth

ミラーレジストリーの base64 でエンコードされたユーザー名およびパスワードを指定します。

<registry>:<port>/<namespace>

ミラーレジストリーの詳細を指定します。

<token>

ミラーレジストリーの base64 でエンコードされた **username:password** を指定します。

以下に例を示します。

```
$ {"auths":{"cloud.openshift.com":
{"auth":"b3BlbnNoaWZ0Y3UjhGOVZPT0IOMEFaUjdPUzRGTA==","email":"user@example.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UjZGOVZPT0IOMEFaUGSTd4VGVGvUjdPUzRGTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZN3VHkxOSTd4VGVGvU1MdTpleUpoYkdjaUailA==","email":"user@example.com"},
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZN3VH3BGSTd4VGVGvU1MdTpleUpoYkdjaU9fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwVWFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1ZXeTRV"},
"my-registry:5000/my-namespace/{\"auth\":\"dXNlcjpyWjAwVWFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1ZXeTRV\"}}}
```

11.4. イメージのミラーリング

クラスターを適切に設定した後に、外部リポジトリからミラーリポジトリにイメージをミラーリングできます。

手順

1. Operator Lifecycle Manager (OLM) イメージをミラーリングします。

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v{product-version}
<mirror_registry>:<port>/olm -a <reg_creds>
```

ここでは、以下ようになります。

product-version

インストールする OpenShift Container Platform のバージョンに対応するタグを指定します (例: **4.8**)。

mirror_registry

Operator コンテンツをミラーリングするターゲットレジストリーおよび namespace の完全修飾ドメイン名 (FQDN) を指定します。ここで、**<namespace>** はレジストリーの既存の namespace です。

reg_creds

変更した **.dockerconfigjson** ファイルの場所を指定します。

以下に例を示します。

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v4.8
mirror.registry.com:443/olm -a ./dockerconfigjson --index-filter-by-os='.*'
```

2. 他の Red Hat が提供する Operator の内容をミラーリングします。

```
$ oc adm catalog mirror <index_image> <mirror_registry>:<port>/<namespace> -a
<reg_creds>
```

ここでは、以下ようになります。

index_image

ミラーリングするカタログのインデックスイメージを指定します。たとえば、これは以前に作成したプルーニングされたインデックスイメージ、または **{index-image-pullspec}** などのデフォルトカタログのソースインデックスイメージのいずれかである可能性があります。

mirror_registry

Operator コンテンツをミラーリングするターゲットレジストリーの FQDN および namespace を指定します。ここで、**<namespace>** はレジストリーの既存の namespace です。

reg_creds

オプション: 必要な場合は、レジストリー認証情報ファイルの場所を指定します。

以下に例を示します。

```
$ oc adm catalog mirror registry.redhat.io/redhat/community-operator-index:v4.8
mirror.registry.com:443/olm -a ./dockerconfigjson --index-filter-by-os='.*'
```

3. OpenShift Container Platform イメージリポジトリをミラーリングします。

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-release:v<product-version>-<architecture> --to=<local_registry>/<local_repository> --to-release-image=<local_registry>/<local_repository>:v<product-version>-<architecture>
```

ここでは、以下のようになります。

product-version

インストールする OpenShift Container Platform のバージョンに対応するタグを指定します (例: **4.8.15-x86_64**)。

architecture

サーバーのアーキテクチャーのタイプを指定します (例: **x86_64**)。

local_registry

ミラーリポジトリのレジストリドメイン名を指定します。

local_repository

レジストリーに作成するリポジトリの名前を指定します (例:**ocp4/openshift4**)。

以下に例を示します。

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-release:4.8.15-x86_64 --to=mirror.registry.com:443/ocp/release --to-release-image=mirror.registry.com:443/ocp/release:4.8.15-x86_64
```

出力例

```
info: Mirroring 109 images to mirror.registry.com/ocp/release ...
mirror.registry.com:443/
  ocp/release
  manifests:
    sha256:086224cadce475029065a0efc5244923f43fb9bb3bb47637e0aaf1f32b9cad47 ->
4.8.15-x86_64-thanos
    sha256:0a214f12737cb1cfbec473cc301aa2c289d4837224c9603e99d1e90fc00328db ->
4.8.15-x86_64-kuryr-controller
    sha256:0cf5fd36ac4b95f9de506623b902118a90ff17a07b663aad5d57c425ca44038c ->
4.8.15-x86_64-pod
    sha256:0d1c356c26d6e5945a488ab2b050b75a8b838fc948a75c0fa13a9084974680cb ->
4.8.15-x86_64-kube-client-agent

.....
sha256:66e37d2532607e6c91eedf23b9600b4db904ce68e92b43c43d5b417ca6c8e63c
mirror.registry.com:443/ocp/release:4.5.41-multus-admission-controller
sha256:d36efdbf8d5b2cbc4dcdbd64297107d88a31ef6b0ec4a39695915c10db4973f1
mirror.registry.com:443/ocp/release:4.5.41-cluster-kube-scheduler-operator
sha256:bd1baa5c8239b23ecdf76819ddb63cd1cd6091119fecdbf1a0db1fb3760321a2
mirror.registry.com:443/ocp/release:4.5.41-aws-machine-controllers
info: Mirroring completed in 2.02s (0B/s)

Success
Update image: mirror.registry.com:443/ocp/release:4.5.41-x86_64
Mirror prefix: mirror.registry.com:443/ocp/release
```

4. 必要に応じて他のレジストリーをミラーリングします。

```
$ oc image mirror <online_registry>/my/image:latest <mirror_registry>
```

関連情報

- Operator カタログのミラーリングについての詳細は、[Mirroring an Operator catalog](#) を参照してください。
- **oc adm catalog mirror** コマンドについての詳細は、[OpenShift CLI administrator command reference](#) を参照してください。

11.5. ミラーレジストリー用のクラスターの設定

イメージを作成し、ミラーレジストリーにミラーリングした後に、Pod がミラーレジストリーからイメージをプルできるようにクラスターを変更する必要があります。

以下を行う必要があります。

- ミラーレジストリー認証情報をグローバルプルシークレットに追加します。
- ミラーレジストリーサーバー証明書をクラスターに追加します。
- ミラーレジストリーをソースレジストリーに関連付ける **ImageContentSourcePolicy** カスタムリソース (ICSP) を作成します。

1. ミラーレジストリー認証情報をクラスターのグローバル pull-secret に追加します。

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=  
<pull_secret_location> 1
```

- 1 新規プルシークレットファイルへのパスを指定します。

以下に例を示します。

```
$ oc set data secret/pull-secret -n openshift-config --from-  
file=.dockerconfigjson=.mirrorsecretconfigjson
```

2. CA 署名のミラーレジストリーサーバー証明書をクラスター内のノードに追加します。

- a. ミラーレジストリーのサーバー証明書が含まれる設定マップを作成します。

```
$ oc create configmap <config_map_name> --from-file=<mirror_address_host>..  
<port>=$path/ca.crt -n openshift-config
```

以下に例を示します。

```
$ oc create configmap registry-config --from-  
file=mirror.registry.com..443=/root/certs/ca-chain.cert.pem -n openshift-config
```

- b. 設定マップを使用して **image.config.openshift.io/cluster** カスタムリソース (CR) を更新します。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。

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

以下に例を示します。

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

3. ICSP を作成し、オンラインレジストリーからミラーレジストリーにコンテナプルリクエストをリダイレクトします。

- a. **ImageContentSourcePolicy** カスタムリソースを作成します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: mirror-ocp
spec:
  repositoryDigestMirrors:
    - mirrors:
      - mirror.registry.com:443/ocp/release ❶
      source: quay.io/openshift-release-dev/ocp-release ❷
    - mirrors:
      - mirror.registry.com:443/ocp/release
      source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

❶ ミラーイメージレジストリーおよびリポジトリの名前を指定します。

❷ ミラーリングされるコンテンツが含まれるオンラインレジストリーおよびリポジトリを指定します。

- b. ICSP オブジェクトを作成します。

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

出力例

```
imagecontentsourcepolicy.operator.openshift.io/mirror-ocp created
```

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。

4. ミラーレジストリーの認証情報、CA、および ICSP が追加されていることを確認します。

- a. ノードにログインします。

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

- b. **/host** をデバッグシェル内のルートディレクトリーとして設定します。

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

- c. **config.json** ファイルで認証情報の有無を確認します。

```
sh-4.4# cat /var/lib/kubelet/config.json
```

出力例

```
{"auths":{"brew.registry.redhat.io":{"xx=="},"brewregistry.stage.redhat.io":  
{"auth":"xxx=="},"mirror.registry.com:443":{"auth":"xx="}}}
```

- 1 ミラーレジストリーおよび認証情報が存在することを確認します。

- d. **certs.d** ディレクトリーに移動します。

```
sh-4.4# cd /etc/docker/certs.d/
```

- e. **certs.d** ディレクトリーの証明書を一覧表示します。

```
sh-4.4# ls
```

出力例

```
image-registry.openshift-image-registry.svc.cluster.local:5000  
image-registry.openshift-image-registry.svc:5000  
mirror.registry.com:443
```

- 1 ミラーレジストリーが一覧にあることを確認します。

- f. ICSP がミラーレジストリーを **registries.conf** ファイルに追加していることを確認します。

```
sh-4.4# cat /etc/containers/registries.conf
```

出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]  
  
[[registry]]  
  prefix = ""  
  location = "quay.io/openshift-release-dev/ocp-release"  
  mirror-by-digest-only = true  
  
[[registry.mirror]]  
  location = "mirror.registry.com:443/ocp/release"  
  
[[registry]]  
  prefix = ""  
  location = "quay.io/openshift-release-dev/ocp-v4.0-art-dev"  
  mirror-by-digest-only = true  
  
[[registry.mirror]]  
  location = "mirror.registry.com:443/ocp/release"
```

registry.mirror パラメーターは、ミラーレジストリーが元のレジストリーの前に検索されることを示します。

g. ノードを終了します。

```
sh-4.4# exit
```

11.6. アプリケーションが引き続き動作することの確認

ネットワークからクラスターを切断する前に、クラスターが想定どおりに機能しており、すべてのアプリケーションが想定どおりに機能していることを確認します。

手順

以下のコマンドを使用して、クラスターのステータスを確認します。

- Pod が実行されていることを確認します。

```
$ oc get pods --all-namespaces
```

出力例

NAMESPACE	STATUS	RESTARTS	AGE	NAME	READY
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-0	
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-1	
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-2	
openshift-apiserver-operator	1/1 Running	3	45m	openshift-apiserver-operator-79c7c646fd-5rvr5	
openshift-apiserver	Running	0	29m	apiserver-b944c4645-q694g	2/2
openshift-apiserver	Running	0	31m	apiserver-b944c4645-shdxb	2/2
openshift-apiserver	Running	0	33m	apiserver-b944c4645-x7rf2	2/2
...					

- ノードが READY のステータスにあることを確認します。

```
$ oc get nodes
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-47ltxb-f76d1-mrffg-master-0	Ready	master	42m	v1.23.0
ci-ln-47ltxb-f76d1-mrffg-master-1	Ready	master	42m	v1.23.0
ci-ln-47ltxb-f76d1-mrffg-master-2	Ready	master	42m	v1.23.0
ci-ln-47ltxb-f76d1-mrffg-worker-a-gsxbz	Ready	worker	35m	v1.23.0
ci-ln-47ltxb-f76d1-mrffg-worker-b-5qqdx	Ready	worker	35m	v1.23.0
ci-ln-47ltxb-f76d1-mrffg-worker-c-rjqpq	Ready	worker	34m	v1.23.0

11.7. ネットワークからクラスターを切断します。

すべての必要なリポジトリをミラーリングし、非接続クラスターとして機能するようにクラスターを設定した後に、ネットワークからクラスターを切断できます。



注記

クラスターがインターネット接続を失うと、Insights Operator のパフォーマンスが低下します。復元できるまで、一時的に [Insights Operator を無効にする](#) ことで、この問題を回避できます。

11.8. パフォーマンスが低下した INSIGHTS OPERATOR の復元

ネットワークからクラスターを切断すると、クラスターのインターネット接続が失われます。Insights Operator は [Red Hat Insights](#) へのアクセスが必要であるため、そのパフォーマンスが低下します。

このトピックでは、Insights Operator をパフォーマンスが低下した状態から復元する方法を説明します。

手順

1. **.dockerconfigjson** ファイルを編集し、**cloud.openshift.com** エントリーを削除します。以下に例を示します。

```
"cloud.openshift.com":{"auth":"<hash>","email":"user@example.com"}
```

2. ファイルを保存します。
3. 編集した **.dockerconfigjson** ファイルでクラスターシークレットを更新します。

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=./.dockerconfigjson
```

4. Insights Operator のパフォーマンスが低下しなくなったことを確認します。

```
$ oc get co insights
```

出力例

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
insights	4.5.41	True	False	False	3d

11.9. ネットワークの復元

非接続クラスターを再接続し、オンラインレジストリーからイメージをプルする場合は、クラスターの ImageContentSourcePolicy (ICSP) オブジェクトを削除します。ICSP がない場合、外部レジストリーへのプルリクエストはミラーレジストリーにリダイレクトされなくなります。

手順

1. クラスターの ICSP オブジェクトを表示します。


```
$ oc get imagecontentsourcepolicy
```

出力例

```
NAME          AGE
mirror-ocp    6d20h
ocp4-index-0  6d18h
qe45-index-0  6d15h
```

2. クラスターの切断時に作成した ICSP オブジェクトをすべて削除します。

```
$ oc delete imagecontentsourcepolicy <icsp_name> <icsp_name> <icsp_name>
```

以下に例を示します。

```
$ oc delete imagecontentsourcepolicy mirror-ocp ocp4-index-0 qe45-index-0
```

出力例

```
imagecontentsourcepolicy.operator.openshift.io "mirror-ocp" deleted
imagecontentsourcepolicy.operator.openshift.io "ocp4-index-0" deleted
imagecontentsourcepolicy.operator.openshift.io "qe45-index-0" deleted
```

3. すべてのノードが再起動して READY ステータスに戻るまで待ち、**registries.conf** ファイルがミラーレジストリーではなく、元のレジストリーを参照していることを確認します。

- a. ノードにログインします。

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

- b. **/host** をデバッグシェル内のルートディレクトリーとして設定します。

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

- c. **registries.conf** ファイルを確認します。

```
sh-4.4# cat /etc/containers/registries.conf
```

出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"] ❶
```

- ❶ 削除した ICSP によって作成された **registry** および **registry.mirror** エントリーが削除されています。

第12章 IBM Z または LINUXONE 環境での追加デバイス設定

OpenShift Container Platform をインストールした後、z/VM でインストールされた IBM Z または LinuxONE 環境でクラスターの追加デバイスを設定できます。次のデバイスを設定できます。

- ファイバーチャネルプロトコル (FCP) ホスト
- FCP LUN
- DASD
- qeth

Machine Config Operator (MCO) を使用し、udev ルールを追加してデバイスを設定するか、デバイスを手動で設定できます。



注記

ここで説明する手順は、z/VM インストールにのみ適用されます。IBM Z または LinuxONE インフラストラクチャーに RHEL KVM を使用してクラスターをインストールした場合、デバイスが KVM ゲストに追加された後、KVM ゲスト内で追加で設定をする必要はありません。ただし、z/VM と RHEL KVM 環境の両方で、Local Storage Operator と Kubernetes NMState Operator を設定する次の手順を適用する必要があります。

関連情報

- [インストール後のマシン設定タスク](#)

12.1. MACHINE CONFIG OPERATOR (MCO) を使用した追加デバイスの設定

このセクションのタスクでは、Machine Config Operator (MCO) の機能を使用して、IBM Z または LinuxONE 環境で追加のデバイスを設定する方法について説明します。MCO を使用したデバイスの設定は永続的ですが、コンピュートノードに対する特定の設定のみを使用できます。MCO では、コントロールプレーンノードに異なる設定を指定できません。

前提条件

- 管理者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。
- z/VM ゲストでデバイスを使用できる必要がある。
- デバイスがすでに接続されている。
- デバイスは、カーネルパラメーターで設定できる **cio_ignore** リストに含まれていない。
- 次の YAML を使用して **MachineConfig** オブジェクトファイルを作成している。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker0
spec:
  machineConfigSelector:
```

```

matchExpressions:
  - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker0]}
nodeSelector:
  matchLabels:
    node-role.kubernetes.io/worker0: ""

```

12.1.1. ファイバーチャネルプロトコル (FCP) ホストの設定

以下は、udev ルールを追加し、N_Port Identifier Virtualization (NPIV) を使用して FCP ホストアダプターを設定する方法の例です。

手順

1. 次の udev ルール **441-zfcp-host-0.0.8000.rules** の例を見てみましょう。

```

ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.8000", DRIVER=="zfcp",
GOTO="cfg_zfcp_host_0.0.8000"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="zfcp", TEST=="[ccw/0.0.8000]",
GOTO="cfg_zfcp_host_0.0.8000"
GOTO="end_zfcp_host_0.0.8000"

LABEL="cfg_zfcp_host_0.0.8000"
ATTR{[ccw/0.0.8000]online}="1"

LABEL="end_zfcp_host_0.0.8000"

```

2. 次のコマンドを実行して、ルールを Base64 エンコードに変換します。

```
$ base64 /path/to/file/
```

3. 以下の MCO サンプルプロファイルを YAML ファイルにコピーします。

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 ❶
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> ❷
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-zfcp-host-0.0.8000.rules ❸

```

- ❶ マシン設定ファイルで定義したロール。
- ❷ 前の手順で生成した Base64 でエンコードされた文字列。

- 3 udev ルールが配置されているパス。

12.1.2. FCP LUN の設定

以下は、udev ルールを追加して FCP LUN を設定する方法の例です。新しい FCP LUN を追加したり、マルチパスで設定済みの LUN にパスを追加したりできます。

手順

1. 次の udev ルール **41-zfcp-lun-0.0.8000:0x500507680d760026:0x00bc000000000000.rules** の例を見てみましょう。

```
ACTION=="add", SUBSYSTEMS=="ccw", KERNELS=="0.0.8000",
GOTO="start_zfcp_lun_0.0.8207"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="start_zfcp_lun_0.0.8000"
SUBSYSTEM=="fc_remote_ports", ATTR{port_name}=="0x500507680d760026",
GOTO="cfg_fc_0.0.8000_0x500507680d760026"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="cfg_fc_0.0.8000_0x500507680d760026"
ATTR{[ccw/0.0.8000]0x500507680d760026/unit_add}="0x00bc000000000000"
GOTO="end_zfcp_lun_0.0.8000"

LABEL="end_zfcp_lun_0.0.8000"
```

2. 次のコマンドを実行して、ルールを Base64 エンコードに変換します。

```
$ base64 /path/to/file/
```

3. 以下の MCO サンプルプロファイルを YAML ファイルにコピーします。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 1
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;base64,<encoded_base64_string> 2
          filesystem: root
          mode: 420
          path: /etc/udev/rules.d/41-zfcp-lun-
            0.0.8000:0x500507680d760026:0x00bc000000000000.rules 3
```

- 1 マシン設定ファイルで定義したロール。

- 2 前の手順で生成した Base64 でエンコードされた文字列。
- 3 udev ルールが配置されているパス。

12.1.3. DASD の設定

以下は、udev ルールを追加して DASD デバイスを設定する方法の例です。

手順

1. 次の udev ルール **41-dasd-eckd-0.0.4444.rules** の例を見てみましょう。

```
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.4444", DRIVER=="dasd-eckd",
GOTO="cfg_dasd_eckd_0.0.4444"
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="dasd-eckd", TEST=="
[ccw/0.0.4444]", GOTO="cfg_dasd_eckd_0.0.4444"
GOTO="end_dasd_eckd_0.0.4444"

LABEL="cfg_dasd_eckd_0.0.4444"
ATTR{[ccw/0.0.4444]online}="1"

LABEL="end_dasd_eckd_0.0.4444"
```

2. 次のコマンドを実行して、ルールを Base64 エンコードに変換します。

```
$ base64 /path/to/file/
```

3. 以下の MCO サンプルプロファイルを YAML ファイルにコピーします。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 1
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,<encoded_base64_string> 2
        filesystem: root
        mode: 420
        path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules 3
```

- 1 マシン設定ファイルで定義したロール。
- 2 前の手順で生成した Base64 でエンコードされた文字列。
- 3 udev ルールが配置されているパス。

12.1.4. qeth の設定

以下は、udev ルールを追加して qeth デバイスを設定する方法の例です。

手順

1. 次の udev ルール **41-qeth-0.0.1000.rules** の例を見てください。

```
ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1001", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1002", DRIVER=="qeth",
GOTO="group_qeth_0.0.1000"
ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.1000", DRIVER=="qeth",
GOTO="cfg_qeth_0.0.1000"
GOTO="end_qeth_0.0.1000"

LABEL="group_qeth_0.0.1000"
TEST=="[ccwgroup/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1000]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1001]", GOTO="end_qeth_0.0.1000"
TEST!="[ccw/0.0.1002]", GOTO="end_qeth_0.0.1000"
ATTR{[drivers/ccwgroup:qeth]group}="0.0.1000,0.0.1001,0.0.1002"
GOTO="end_qeth_0.0.1000"

LABEL="cfg_qeth_0.0.1000"
ATTR{[ccwgroup/0.0.1000]online}="1"

LABEL="end_qeth_0.0.1000"
```

2. 次のコマンドを実行して、ルールを Base64 エンコードに変換します。

```
$ base64 /path/to/file/
```

3. 以下の MCO サンプルプロファイルを YAML ファイルにコピーします。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker0 1
  name: 99-worker0-devices
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;base64,<encoded_base64_string> 2
```

```
filesystem: root
mode: 420
path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules 3
```

- 1 マシン設定ファイルで定義したロール。
- 2 前の手順で生成した Base64 でエンコードされた文字列。
- 3 udev ルールが配置されているパス。

次のステップ

- [Local Storage Operator \(LSO\) のインストールおよび設定](#)
- [ノードのネットワーク設定の更新](#)

12.2. 追加のデバイスの手動設定

このセクションのタスクでは、IBM Z または LinuxONE 環境で追加のデバイスを手動で設定する方法について説明します。この設定方法はノードの再起動後も持続しますが、OpenShift Container Platform ネイティブではなく、ノードを置き換える場合は手順をやり直す必要があります。

前提条件

- 管理者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。
- デバイスがノードで使用可能である。
- z/VM 環境では、デバイスを z/VM ゲストに接続しておく。

手順

1. 次のコマンドを実行して、SSH 経由でノードに接続します。

```
$ ssh <user>@<node_ip_address>
```

次のコマンドを実行して、ノードへのデバッグセッションを開始することもできます。

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

2. **chzdev** コマンドでデバイスを有効にするには、次のコマンドを入力します。

```
$ sudo chzdev -e 0.0.8000
sudo chzdev -e 1000-1002
sude chzdev -e 4444
sudo chzdev -e 0.0.8000:0x500507680d760026:0x00bc000000000000
```

関連情報

IBM ドキュメントの [永続的なデバイス設定](#) を参照してください。

12.3. ROCE ネットワークカード

RoCE (RDMA over Converged Ethernet) ネットワークカードは、有効にする必要はなく、ノードで使用できる場合はいつでも Kubernetes NMState Operator で設定できます。たとえば、RoCE ネットワークカードは、z/VM 環境に接続されているか、RHEL KVM 環境でパススルーされている場合に使用できます。

12.4. FCP LUN のマルチパスの有効化

このセクションのタスクでは、IBM Z または LinuxONE 環境で追加のデバイスを手動で設定する方法について説明します。この設定方法はノードの再起動後も持続しますが、OpenShift Container Platform ネイティブではなく、ノードを置き換える場合は手順をやり直す必要があります。



重要

IBM Z および LinuxONE では、インストール時にクラスターを設定した場合のみマルチパスを有効にできます。詳細は、**IBM Z および LinuxONE への z/VM を使用したクラスターのインストール**の RHCOS のインストールおよび OpenShift Container Platform ブートストラッププロセスの開始を参照してください。

前提条件

- 管理者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。
- 上記で説明したいずれかの方法で、LUN への複数のパスを設定している。

手順

1. 次のコマンドを実行して、SSH 経由でノードに接続します。

```
$ ssh <user>@<node_ip_address>
```

次のコマンドを実行して、ノードへのデバッグセッションを開始することもできます。

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

2. マルチパスを有効にするには、次のコマンドを実行します。

```
$ sudo /sbin/mpathconf --enable
```

3. **multipathd** デーモンを開始するには、次のコマンドを実行します。

```
$ sudo multipath
```

4. オプション: マルチパスデバイスを fdisk でフォーマットするには、次のコマンドを実行します。

```
$ sudo fdisk /dev/mapper/mpatha
```

検証

- デバイスがグループ化されたことを確認するには、次のコマンドを実行します。

```
$ sudo multipath -ll
```


出力例

```
mpatha (20017380030290197) dm-1 IBM,2810XIV
  size=512G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
-- policy='service-time 0' prio=50 status=enabled
|- 1:0:0:6 sde 68:16 active ready running
|- 1:0:1:6 sdf 69:24 active ready running
|- 0:0:0:6 sdg 8:80 active ready running
`- 0:0:1:6 sdh 66:48 active ready running
```

次のステップ

- [Local Storage Operator \(LSO\) のインストールおよび設定](#)
- [ノードのネットワーク設定の更新](#)